Protocol API

Open Modbus/TCP

V2.6.0

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC071103API10EN | Revision 10 | English | 2016-08 | Released | Public

Table of contents

1 Introduction 4
1.1 Abstract 4
1.2 List of revisions 4
1.3 Functional overview 5
1.4 System requirements 5
1.5 Intended audience 5
1.6 Technical data 6
1.7 Terms, abbreviations and definitions 8
1.8 References to documents 8
1.9 Legal notes 9
1.9.1 Copyright 9
1.9.2 Important notes 9
1.9.3 Exclusion of liability 10
1.9.4 Export 10

2 Getting started 11
2.1 Client/server and operating modes 11
2.2 Structure of the Open Modbus/TCP protocol stack 12
2.3 Configuration 14

3 Exchanging data 15
3.1 Message mode vs IO mode 15
3.2 Command table 16
3.3 Modbus function codes 16
3.3.1 Function Code 01 (0x01) Read Coils 17
3.3.2 Function Code 02 (0x02) Read Input Discretes 18
3.3.3 Function Code 03 (0x03) Read Multiple Registers 19
3.3.4 Function Code 04 (0x04) Read Input Registers 20
3.3.5 Function Code 05 (0x05) Write Coils 21
3.3.6 Function Code 06 (0x06) Write Single Register 22
3.3.7 Function Code 07 (0x07) Read Exception Status 23
3.3.8 Function Code 15 (0x0F) Force Multiple Coils 24
3.3.9 Function Code 16 (0x10) Write Multiple Registers 25
3.3.10 Function Code 23 (0x17) Read/Write Multiple Registers 26
3.3.11 Function Code 43 (0x2B) Read Device Identification 27
3.3.12 Troubleshooting 28

4 The application interface 29
4.1 Configuration 30
4.1.1 Configuration via packet 30
4.1.2 Configuration via config file inibatch.nxd 32
4.1.3 Configuration of the command table 32
4.1.4 OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ/CNF – Set configuration 34
4.1.5 Confirmation 44
4.2 Common RCX packets 45
4.3 IO mode: Server Modbus data model 47
4.4 Message mode: Server with packets 50
4.4.1 Overview 50
4.4.2 OMB_OMBTASK_CMD_RECEIVE_IND/RES – Receive data indication 50
4.4.3 OMB_OMBTASK_CMD_CONNECTION_IND/RES – Connection indication 59
4.5 Message mode: Client with packets 62
4.5.1 Overview 62
4.5.2 OMB_OMBTASK_CMD_SEND_REQ/CNF - Send data request 62
4.6 Message mode: Client with command table 72
4.6.1 Overview 72
4.6.2 CMDTBL_INIT_REQ/CNF – Init command table task 74
4.6.3 CMDTBL_ADD_TABLE_REQ/CNF – Add new command table 76
4.6.4 CMDTBL_DELETE_TABLE_REQ/CNF – Delete a command table 78
4.6.5 CMDTBL_ACTIVATE_TABLE_REQ/CNF – Activate a command table 80
4.6.6 CMDTBL_DEACTIVATE_TABLE_REQ/CNF – Activate a command table 82
4.6.7 CMDTBL_ADD_COMMAND_REQ/CNF – Add a command to a table 84
4.6.8 CMDTBL_DELETE_COMMAND _REQ/CNF – Delete a command 87
4.6.9 CMDTBL_ACTIVATE_COMMAND_REQ/CNF – Activate a command 89

4.6.10 CMDTBL_DEACTIVATE_COMMAND_REQ/CNF – Deactivate a command 91
4.6.11 CMDTBL_TRIGGER_COMMAND_REQ/CNF – Trigger a command 93
4.6.12 CMDTBL_GET_IO_INFO_REQ/CNF – Get IO Info 95
4.6.13 CMDTBL_DEINIT_REQ/CNF – Table Deinitialization 97
4.6.14 CMDTBL_START_STOP_REQ/CNF – Table Start/Stop 98
4.6.15 Command Table diagnostic with Status bit fields 99
4.6.16 Command table timing diagram 101

5 Special topics 103
5.1 For programmers 103
5.2 Task start-up parameters 103
5.2.1 Start-up parameters of the OMB task 103
5.2.2 Start-up parameters of the OMB_AP task 105
5.2.3 Start-up parameters of the CMD task 106

6 Status/Error codes 107
6.1 Status/Error codes OMB task 107
6.2 Status/Error codes OMB_AP task 111
6.3 Status/Error codes CMD task 112

7 Appendix 114
7.1 List of tables 114
7.2 List of figures 115
7.3 Contacts 116

1 Introduction

1.1 Abstract

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

1.2 List of revisions

Rev	Date	Name	Revisions
8	2013-09-24	RG	Firmware/stack version 2.5.8.x Reference to netX Dual-Port Memory Interface Manual Revision 12 Additional remark at parameter description. New error message description added.
9	2016-06-07	MK	Firmware/stack version 2.5.20.x Add new packet Connection Indication Add new configuration flags Chg allow UnitId's from 0 … 255
10	2016-08-01	MK, HH	Firmware/stack version 2.6.0
			Document restructured.
			Section *Client/server and operating modes* added.
			Section *Command table* added.
			Section *Extended parameter* added.
			Section *Ident parameter* added.
			Section *Message mode: Client with command table* added.
			Section *Special topics* added.

Table 1: List of revisions

1.3 Functional overview

The stack has been written in order to meet the Open Modbus/TCP protocol specification. The stack can operate a client or as a server.

The main functionality from the application view is:

- Message mode (Client and Server)
- IO mode (Server)
- Command table execution (Client)

1.4 System requirements

This software package has following system requirements to its environment:

- netX chip as hardware platform
- rcX operating system for task scheduling required

1.5 Intended audience

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

- Knowledge of the programming language C
- Knowledge of the use of the real-time operating system rcX
- Knowledge of the Hilscher Task Layer Reference Model
- Knowledge of the TCP/IP protocol suite
- Knowledge of the Open Modbus/TCP protocol

1.6 Technical data

The data below applies to Open Modbus/TCP firmware and stack version V2.6.0.

The firmware/stack is based on the MODBUS specifications references in [3] and [4].

Technical data

Feature	Value
Maximum number of input data	5760 bytes
Maximum number of output data	5760 bytes
Acyclic communication	Read/Write register: Max. 125 registers per read telegram (FC 3, 4, 23) Max. 121 registers per write telegram (FC 23), Max. 123 registers per write telegram (FC 16)
	Read/Write coil: Max. 2000 coils per read telegram (FC 1, 2) Max. 1968 coils per write telegram (FC 15)
Modbus function codes	1, 2, 3, 4, 5, 6, 7, 15, 16, 23, 43
Modes	Message Mode: Client, Server (I/O data area is not used in this mode) I/O Mode: Server
Command table	Max. 16 servers configurable Max. 256 commands in sum *)
Baud rates	10 and 100 MBit/s
Data transport layer	Ethernet II, IEEE 802.3

Table 2: Technical data Open Modbus/TCP

Firmware/stack available for netX

netX	Available
netX 50	Yes
netX 51	Yes
netX 52	Yes
netX 100	Yes
netX 500	Yes

Table 3: Firmware/stack available for netX

Configuration

Configured by	Remark
SYCON.net	Download or by exported configuration file named inibatch.nxd and command.nxd
netX Configuration Tool	Download (netX Configuration Tool doesn't configure the command table)
Application	Application can use packets to configure the firmware/stack.

Table 4: Configuration

Diagnostic

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

Additional features

Feature	Value
DMA transfer	Supported for PCI target
Slot number	Supported for CIFX 50-RE, CIFX 50E-RE
Integrated WebServer	Supported (for details see reference [6])

Table 5: Additional features

Limitations

*) For the netRAPID device only 64 commands can be configured.

1.7 Terms, abbreviations and definitions

Term	Description
AP (task)	Application (task) on top of the stack
BOOTP	Bootstrap Protocol
DHCP	Dynamic Host Configuration Protocol
DPM	Dual-port memory
IP	Internet Protocol
OMB	Open Modbus/TCP
TCP	Transmission Control Protocol
UDP	User Datagram Protocol
CMD	Command Table

Table 6: 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.

All IP addresses in this document have host byte order.

1.8 References to documents

This document is based on the following specifications:

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

[2] Hilscher Gesellschaft für Systemautomation mbH: TCP/IP Protocol Interface Manual, Revision 13, English, 2015

[3] MODBUS APPLICATION PROTOCOL SPECIFICATION, V1.1a, June 4, (http://www.modbus.org)

[4] MODBUS MESSAGING ON TCP/IP IMPLEMENTATION GUIDE, V1.0a, June 4, 2004 (http://www.modbus.org)

[5] Hilscher Gesellschaft für Systemautomation mbH: „Ethernet Protocol API.pdf“ Rev05

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

1.9 Legal notes

1.9.1 Copyright

1.9.2 Important notes

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

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

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

1.9.3 Exclusion of liability

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

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

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

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

1.9.4 Export

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

2 Getting started

This section explains some essential information you should know when starting to work with the Open Modbus/TCP protocol API.

2.1 Client/server and operating modes

Open Modbus/TCP is a TCP/IP-based communication. One device operates as client and requests data from another device which operates as a server. In a request, the client uses a function code to specify the function which the server should execute. The client can read and write data memory of the server.

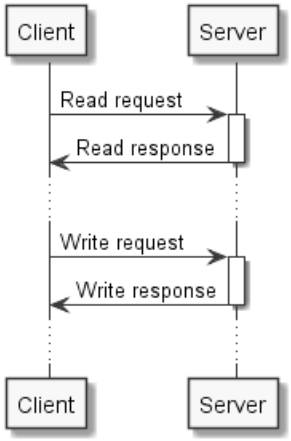


Figure 1: Client/Server principle

Open Modbus/TCP offers two data types:

- register (16-bit value)
- coils (1-bit value)

The Open Modbus/TCP task offers two operation modes:

- Message mode: stack operates as client only, as server only or as client and server
- IO mode: stack operates as server (only)

2.2 Structure of the Open Modbus/TCP protocol stack

The figure below shows the internal structure of the tasks which together represent the Open Modbus/TCP stack.

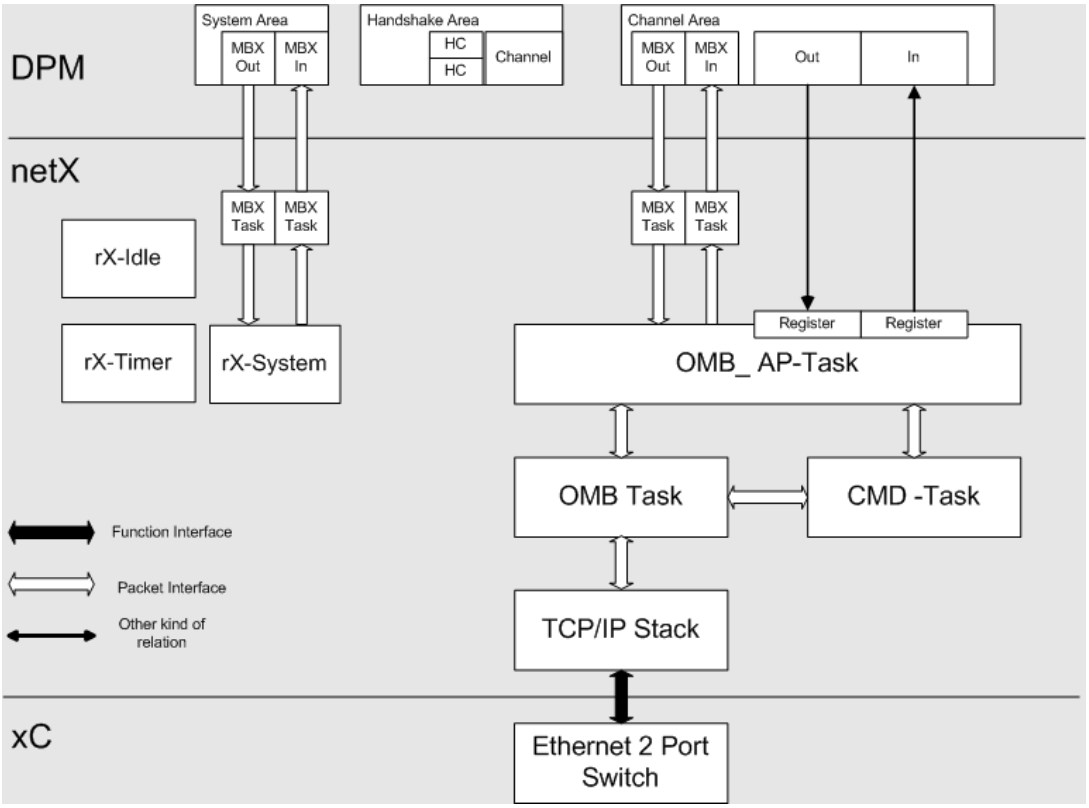


Figure 2: Structure of Open Modbus/TCP Firmware

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

The user application only accesses the task located in the highest layer namely the AP task which constitutes the application interface of the Open Modbus/TCP protocol stack.

OMB_AP task

The OMB_AP task provides the interface to the user application and the control of the stack. It also completely handles the dual-port memory interface of the communication channel. In detail, it is responsible for the following:

- Handling the communication channels dual-port memory interface
  - Process data exchange (IO mode)
  - Channel mailboxes
  - Watchdog
  - Service the common and extended status field in the dual-port memory
- Configuration management
  - Loading the data base file ‘command.nxd’
  - Reception of configuration packets provided by data base file ‘inibatch.nxd’
  - Reception and routing of configuration packets send by the user application
- Mailbox packet handling and routing
  - handle of all incoming and outgoing packets to the user application
  - configuration packets
  - send/receive packets
  - handle Hilscher common RCX stack packets like `RCX_CHANNEL_INIT_REQ`
  - forward ‘Request’ and ‘Response’ packets from host application context through the OMB task
  - forward ‘Indication’ and ‘Confirmation’ packets from the OMB task through the host application context
  - forward ‘Request’ and ‘Response’ packets from host application context through the CMD task
  - forward ‘Request’ and ‘Response’ packets from host application context through the TCP/IP task
- Control of LEDs

OMB task

The OMB task is the Open Modbus/TCP stack implementation. It manages the conversion of Modbus functions to TCP/IP frames. It represents the central part of the Open Modbus/TCP stack implementation. It is responsible for the protocol handling, the communication to/from TCP/IP stack and it is the counterpart of the OMB_AP task.

The underlying TCP/IP stack provides basic TCP/IP communication capabilities located at layers 3 and 4 of the OSI/ISO layer model.

The OMB task is the Open Modbus/TCP stack implementation. It is responsible for the protocol handling, the communication from and to the TCP/IP stack and it is the counterpart of the OMB_AP task.

CMD task

The CMD task (Command Table) is the executer of a configured command table. The command table is a list of configured Open Modbus/TCP request that are executed cyclic. The command table works as Open Modbus/TCP client. It can send reading or writing function codes to a remote Open Modbus/TCP server. The data that is read or written from a remote Open Modbus/TCP server are exchanged with the host application via dual-port memory input/output areas.

2.3 Configuration

The Open Modbus/TCP stack requires configuration parameters e.g. IP address.

Configuration parameters can be set using configuration software or can be set from an application program using the packet API.

3 Exchanging data

3.1 Message mode vs IO mode

The Open Modbus/TCP task offers two basic operation modes:

- Message mode: stack operates as client only, as server only or as client and server
- IO mode: stack operates as server (only)

The parameter `ulMode` sets the operation mode of the Open Modbus/TCP stack. This parameter is one of the basic configuration parameters which are described in section *Basic parameter* on page 38.

Message mode

In message mode the stack uses packets to communicate to the application. In this mode, a simultaneous operation of the Open Modbus/TCP task as server and client is possible.

The Open Modbus/TCP stack can handle up to 16 sockets to establish 16 TCP/IP connections simultaneously. By default 4 sockets are configured as server sockets and the 12 sockets can be used as client sockets. The proportion of the available 16 sockets (server sockets to client sockets) can be parameterized. With the default setting 4 clients can connect to the stack over TCP/IP. The rest of 12 sockets can be used from a client application to send commands to different Open Modbus/TCP devices or Open Modbus/TCP application.

In message mode the application programmer has the option to implement its own Modbus data model. The application programmer can decide which function codes are supported. It is also possible to support the full Modbus address range of Modbus register and coils.

As an alternative, a command table can be used instead of using packets. In this case, the stack automatically executes the commands of the command table to act as a client and request data from Modbus server. The stack uses the input and output area of the dual-port memory to communicate to the application. The application program cannot use packets when using the command table is used.

IO mode

In IO mode the stack uses the input and output area of the dual-port memory to communicate to the application. In IO mode the Open Modbus/TCP task works exclusive as server (No client functionality is available).

In IO mode the Modbus data model is fixed. The entire handling of Open Modbus/TCP function codes is done by the Open Modbus/TCP stack. The number of supported register and coils is limited by the size of the dual-port input and output area.

3.2 Command table

When the Open Modbus/TCP stack is configured in Message mode, it supports a so called **command table**. The command table contains a list commands which contain function codes, addresses and further parameters. These commands are processed cyclically by the Modbus/TCP stack. The command table can contain several function codes which are sent to one or different Open Modbus/TCP servers. The command table is a configuration file which is created by SYCON.net or netX Configuration Tool and downloaded to the Open Modbus/TCP task during the configuration process. Since Modbus/TCP stack version V2.6 it is also possible for the host application to configure a command table using packets.

When the command table is processed, the data exchange with the host application is done via dual-port memory input and output area.

When the Modbus/TCP stack executes a reading function code from the command table then the data which are read from the remote Modbus/TCP server will be placed into the input image of the dual-port memory. When the Modbus/TCP stack executes a writing function code from the command table then the data which are written to the remote Modbus/TCP server will be taken from the output image of the dual-port memory.

3.3 Modbus function codes

The Modbus specification defines the function codes for send and receive operations:

Function code	Value (decimal)	Function
FC1	1	Read coils
FC2	2	Read input discretes
FC3	3	Read multiple registers
FC4	4	Read input registers
FC5	5	Write coil
FC6	6	Write single register
FC7	7	Read exception status
FC15	15	Force multiple coils
FC16	16	Write multiple registers
FC23	23	Read/Write multiple registers
FC43	43	Read Device Identification

Table 7: Function codes

3.3.1 Function Code 01 (0x01) Read Coils

Coils in the sense of Open Modbus/TCP are status bits which can be both read and written.

This function code can be used to read from 1 up to 2000 status values of coils in a remote device. The addresses of the coils must be contiguous.

There are two parameters required for processing this command:

1. the starting address `tFcStd.ulDataAdr`, i.e. the address of the first coil specified,
2. and the number of coils to be read out `tFcStd.ulDataCnt`.

The response needs to contain the following parameters:

- The number of bytes which are delivered (depending of the number of coils requested in the receive
- The contents of the coils.

The LSB of the first byte should contain the requested output data. The rest of the coils follow in direction to the high order end of this byte, and in ascending order (i.e. from the low order end to high order end) within the subsequent bytes.

According to the Modbus standard, each coil in the response message is represented as one bit of the data field.

Status should be indicated as

- 1= ON and
- 0= OFF.

Coils are addressed starting with the value 0 at the first coil. Thus the coils #1-65536 are addressed as with the values 0-65535.

To write coils use function codes 05 or 15, see below.

Example

If you want to read 10 coils at reference 100, use the following values:

Variable	Value
`ulRouting`	192.168.10.16 `(= 0xC0A80A10)`
`ulUnitId`	0
`ulFunctionCode`	1 (“Read Coils”)
`ulException`	0
`tFcStd.ulDataAdr`	100 (“Offset 100”)
`tFcStd.ulDataCnt`	10 (“10 bits”)

Table 8: Example FC 01

3.3.2 Function Code 02 (0x02) Read Input Discretes

This function code can be used to read from 1 up to 2000 discrete status values of read-only data in a remote device. The addresses must be contiguous.

There are two parameters required for processing this command:

1. the reference number `tFcStd.ulDataAdr` i.e. the address of the first coil specified,
2. and the number of bits to be read out `tFcStd.ulDataCnt`.

The response needs to contain the following parameters:

- The number of bytes which are delivered (depending of the number of bits requested)
- The values of the bits.

The LSB of the first byte should contain the requested output data for the first requested bit. The rest of the bit values follow in direction to the high order end of this byte, and in ascending order (i.e. from the low order end to high order end) within the subsequent bytes.

Status should be indicated as

- 1 = ON
- 0 = OFF

Bits are addressed starting with the value 0 at the first coil. Thus the bits #1-65536 are addressed as with the values 0-65535.

Example

If you want to read 12 discrete inputs at reference 280, use the following values:

Variable	Value
`ulRouting`	`192.168.10.16 (= 0xC0A80A10)`
`ulUnitId`	0
`ulFunctionCode`	2 (“Read Input Discretes”)
`ulException`	0
`tFcStd.ulDataAdr`	280 (“Offset 280”)
`tFcStd.ulDataCnt`	12 (“12 bit”)

Table 9: Example FC 02

3.3.3 Function Code 03 (0x03) Read Multiple Registers

This function code can be used to read from 1 up to 125 16-bit registers containing read-/write data from a remote device. The addresses of the registers must be contiguous.

There are two parameters required for processing this command:

1. the reference number where to start, `tFcStd.ulDataAdr`, i.e. the address of the first register specified,
2. and the number of registers to be read out (`tFcStd.ulDataCnt`).

The response needs to contain the following parameters:

- The number of bytes which are delivered (actually twice the number of registers to be read out specified in the request)
- The values of the requested registers.

Registers are addressed starting with the value 0 at the first register. Thus the registers #1-65536 are addressed as with the values 0-65535.

Example

If you want to read 100 registers at reference 400, use the following values:

Variable	Value
`ulRouting`	192.168.10.16 (= `0xC0A80A10`)
`ulUnitId`	0
`ulFunctionCode`	3 (“Read Multiple Register”)
`ulException`	0
`tFcStd.ulDataAdr`	400 (“Offset 400”)
`tFcStd.ulDataCnt`	100 (“100 Registers”)

Table 10: Example FC 03

To write registers use function codes 06 or 16, see below.

3.3.4 Function Code 04 (0x04) Read Input Registers

This function code can be used to read from 1 up to 125 16-bit registers containing read-only data from a remote device. The addresses of the registers must be contiguous.

There are two parameters required for processing this command:

1. the reference number `tFcStd.ulDataAdr`, i.e. the address of the first register specified,
2. and the number of registers to be read out (`tFcStd.ulDataCnt`).

The response needs to contain the following parameters:

- The number of bytes which are delivered (actually twice the number of registers to be read out specified in the request)
- The values of the registers.

Registers are addressed starting with the value 0 at the first register. Thus the registers #1-65536 are addressed as with the values 0-65535.

Example

If you want to read 1 input register at reference 0, use the following values:

Variable	Value
`ulRouting`	192.168.10.16 (= `0xC0A80A10`)
`ulUnitId`	0
`ulFunctionCode`	4 (“Read Input Registers”)
`ulException`	0
`tFcStd.ulDataAdr`	0 (“Offset 0”)
`tFcStd.ulDataCnt`	1 (“1 register”)

Table 11: Example FC 04

3.3.5 Function Code 05 (0x05) Write Coils

Coils in the sense of Open Modbus/TCP are status bits which can be read and written.

This function code can be used to write one status value of a single coil within a remote device.

There are two parameters required for processing this command:

1. the address `tFcStd.ulDataAdr` of the coil specified,
2. and the value of the coil to be written (0xFF to set the bit and 0x00 to clear the bit).

The response needs to contain the following parameters:

- The address of the coil specified
- and the value written to the coil (as repetition).

Coils are addressed starting with the value 0 at the first coil. Thus the coils #1-65536 are addressed as with the values 0-65535.

Example

If you want to write 1 coil at reference 0 to the value 1, use the following values:

Variable	Value
`ulRouting`	`192.168.10.16 (= 0xC0A80A10)`
`ulUnitId`	0
`ulFunctionCode`	5 (“Write coil”)
`ulException`	0
`tFcStd.ulDataAdr`	0 (“Offset 0”)
`tFcStd.ulDataCnt`	1 (“1 bit”)
`tFcStd.abData[0]`	0xFF

Table 12: Example FC 05

3.3.6 Function Code 06 (0x06) Write Single Register

This function code can be used to write one 16-bit register containing read-/write data within a remote device.

There are two parameters required for processing this command:

1. the address `tFcStd.ulDataAdr` of the register specified,
2. and the 16-bit value of the register to be written.

The response needs to contain the following parameters:

- the address of the register specified,
- and the 16-bit value of the register to be written (as repetition).

Registers are addressed starting with the value 0 at the first register. Thus the registers #1-65536 are addressed as with the values 0-65535.

Example

If you want to write one register at reference 8 of value 0x1234, use the following values:

Variable	Value
`ulRouting`	`192.168.10.16 (= 0xC0A80A10)`
`ulUnitId`	0
`ulFunctionCode`	6 (“Write single register”)
`ulException`	0
`tFcStd.ulDataAdr`	8 (“Offset 8”)
`tFcStd.ulDataCnt`	1 (“1 Register”)
`abData[0]`	0x12 (see Note below)
`abData[1]`	0x34 (see Note below)

Table 13: Example FC 06

Note: Consider also the parameter *Swap.*

3.3.7 Function Code 07 (0x07) Read Exception Status

This function code can be used to read the Exception Status of a remote device. Except the IP address, there are no further parameters needed.

Example

If you want to read the Exception status, use the following values:

Variable	Value
`ulRouting`	`192.168.10.16 (= 0xC0A80A10)`
`ulUnitId`	0
`ulFunctionCode`	7 (“Read Exception status”)
`ulException`	0
`tFcStd.ulDataAdr`	0
`tFcStd.ulDataCnt`	0

Table 14: Example FC 07

The response contains the Exception status from remote station in `tFcStd.abData[0]` (1 byte = eight Exception status outputs).

The Exception status of this Open Modbus/TCP stack is:

Bits	Name (Bit mask)	Description
7 ... 3	Reserved	Reserved for future use
2	`OMB_OMBTASK_COIL_FC7_ NO_IO_ACCESS`	Access to IO Data Image: If set, the access to the Input/Output Data Image in the dual-port memory is blocked (not implemented!). Modbus requests from the line are rejected with Exception code 07.
1	`OMB_OMBTASK_COIL_FC7_ WATCHDOG_ERROR`	Watchdog: If set, a watchdog error is active. Modbus requests from the line are rejected with Exception code 07.
0	`OMB_OMBTASK_COIL_FC7_ USER_NOT_READY`	Host application: If set, the host application is not registered. In Message mode, there is no communication possible. Modbus requests from the line are rejected with Exception code 07.

Table 15: Exception status

Note: Per default in Server Mode the Modbus request with FC07 is handled by the stack internally and responded to the client immediately. No indication to the host application is generated.

Since Open Modbus/TCP stack V2.6 it is possible to forward FC07 to the host application. But this must be enabled by the corresponding configuration flag. In this case the meaning of the exception bits are user specific.

3.3.8 Function Code 15 (0x0F) Force Multiple Coils

Coils in the sense of Open Modbus/TCP are status bits which can be read and written.

This function code can be used to write status values of multiple coils at once to a remote device.

The applicable range extends from 1 to 1968.

There are the following parameters required for processing this command:

1. the reference number of the coil where to start (`tFcStd.ulDataAdr`),
2. the number of coils (bits) to be written (`tFcStd.ulDataCnt`)
3. and the values of the coils to be written where the least significant bit represents first coil).

The response needs to contain the following parameters:

- The address `tFcStd.ulDataAdr` of the coil specified
- and the number of coils forced.

Coils are addressed starting with the value 0 at the first coil. Thus the coils #1-65536 are addressed as with the values 0-65535.

Example

If you want to write 3 coils (Numbers 1, 2, 3) at reference 0 to values 0, 0, 1, use the following values:

Variable	Value
`ulRouting`	192.168.10.16 (= 0xC0A80A10)
`ulUnitId`	0
`ulFunctionCode`	15 (“Force Multiple Coils”)
`ulException`	0
`tFcStd.ulDataAdr`	0 (“Offset 0”)
`tFcStd.ulDataCnt`	3 (“3 bits”)
`tFcStd.abData[0]`	0x04

Table 16: Example FC 15

3.3.9 Function Code 16 (0x10) Write Multiple Registers

This function code can be used to write to 1 up to 123 16-bit registers containing read-/write data at a remote device. The addresses of the registers must be contiguous.

The following parameters are required for processing this command:

1. the reference number where to start (`tFcStd.ulDataAdr`)
2. the number of registers to write (`tFcStd.ulDataCnt`).
3. and the values to write to the registers

The response needs to contain the following parameters:

- The specified reference number
- the number of registers that has been written

Registers are addressed starting with the value 0 at the first register. Thus the registers #1-65536 are addressed as with the values 0-65535.

Example

If you want to write 1 register at reference 20000 of value 0x1234, use the following values:

Variable	Value
`ulRouting`	192.168.10.16 (= `0xC0A80A10`)
`ulUnitId`	0
`ulFunctionCode`	16 (“Write multiple registers”)
`ulException`	0
`tFcStd.ulDataAdr`	20000 (“Offset 20000”)
`tFcStd.ulDataCnt`	1 (“1 Register”)
`tFcStd.abData[0]`	0x12
`tFcStd.abData[1]`	0x34

Table 17: Example FC 16

Note: Consider also the parameter *Swap.*

3.3.10 Function Code 23 (0x17) Read/Write Multiple Registers

This function code can be used to perform a combined read/write of 16-bit register in a single Modbus transaction. It can

- write to 1 up to 121 16-bit registers
- read to 1 up to 125 16-bit registers

containing read/write data at a remote device. The addresses of the registers must be contiguous.

The following parameters are required for processing this command:

1. the reference number where to start reading (`tFc23.ulDataAdrRead`)
2. the number of registers to read (`tFc23.ulDataCntRead`)
3. the reference number where to start writing (`tFc23.ulDataAdrWrite`)
4. the number of registers to write (`tFc23.ulDataCntWrite`)
5. and the values to write to the registers (starting at `tFc23.abData[0]`)

The response contains the read data, starting at `tFc23.abData[0]`.

Registers are addressed starting with the value 0 at the first register. Thus the registers #1-65536 are addressed as with the values 0-65535.

Example

If you want to read 10 registers starting at reference 100 and write 2 registers starting at reference 200 of values 0x1234, 0x5678, use the following values:

Variable	Value
`ulRouting`	192.168.10.16 (= `0xC0A80A10`)
`ulUnitId`	0
`ulFunctionCode`	23 (“Read/Write multiple registers”)
`ulException`	0
`tFc23.ulDataAdrRead`	100 (“Offset 100”)
`tFc23.ulDataCntRead`	10 (“10 Registers”)
`tFc23.ulDataAdrWrite`	200 (“Offset 200”)
`tFc23.ulDataCntWrite`	2 (“2 Registers”)
`tFc23.abData[0]`	0x12
`tFc23.abData[1]`	0x34
`tFc23.abData[2]`	0x56
`tFc23.abData[3]`	0x78

Table 18: Example FC 23

3.3.11 Function Code 43 (0x2B) Read Device Identification

This function code allows reading the identification and additional information relative to the physical and functional description of a remote device. The Read Device Identification interface is modeled as an address space composed of a set of addressable data elements. The data elements are called objects and an object Id identifies them.

The interface consists of 3 categories of objects:

- **Basic Device Identification.** All objects of this category are mandatory: VendorName, Product code, and revision number.
- **Regular Device Identification**. In addition to Basic data objects, the device provides additional and optional identification and description data objects. All of the objects of this category are defined in the standard but their implementation is optional.
- **Extended Device Identification.** In addition to regular data objects, the device provides additional and optional identification and description private data about the physical device itself. All of these data are device dependent.

Note: Function Code 43 is supported beginning with Modbus/TCP Stack V2.6. The Device Identification data are configured with the Set Configuration packet. The Modbus/TCP Stack V2.6 supports the “Basic” and “Regular” Device Identification. The “Conformity Level” 0x82 is supported. This means the Device Identification data can be accessed per “stream” or “individual”. The “Extended” Device Identification is not supported for loadable firmware. But it is optionally supported, in case of custom specific firmware based on linkable objects. In this case the Extended Device Identification data are configured as task start up parameter.

Note: Function Code 43 is only supported as server. This means the Modbus/TCP stack will respond to FC43 request from a client. Sending FC43 to remote server as client is not supported.

3.3.12 Troubleshooting

When using these function codes, the following error situations might occur:

Illegal Function

The function code received in the query is not allowed (Server mode). This could be

- because the function code does not belong to the allowed range (currently 1 … 7, 15, 16 or 23).
- because the server is not in the correct state to process such a request, for example because it has not been configured correctly and is asked to return register values.

These situations might lead to an error message `Exception code ILLEGAL FUNCTION (Code 01)`.

Other possibilities include:

- Wrong number of data in connection of the Modbus reference number
- Wrong data address

These situations might lead to an error message `Exception code ILLEGAL DATA ADDRESS (Code 02)`.

Illegal Data Address

The data address received in the query is not a correct address for the slave, i.e. the combination of reference number and transfer length is invalid. For a controller with 100 registers, a request with offset 96 and length 4 could be executed successfully, while a request with offset 96 and length 5 will cause this error.

In IO mode, this situation might lead to an error message `TLR_E_OMB_OMBTASK_MOD_MEM_MOD_START_ADR (0xC0600004)`.

4 The application interface

This chapter describes the application interface of the Open Modbus/TCP stack.

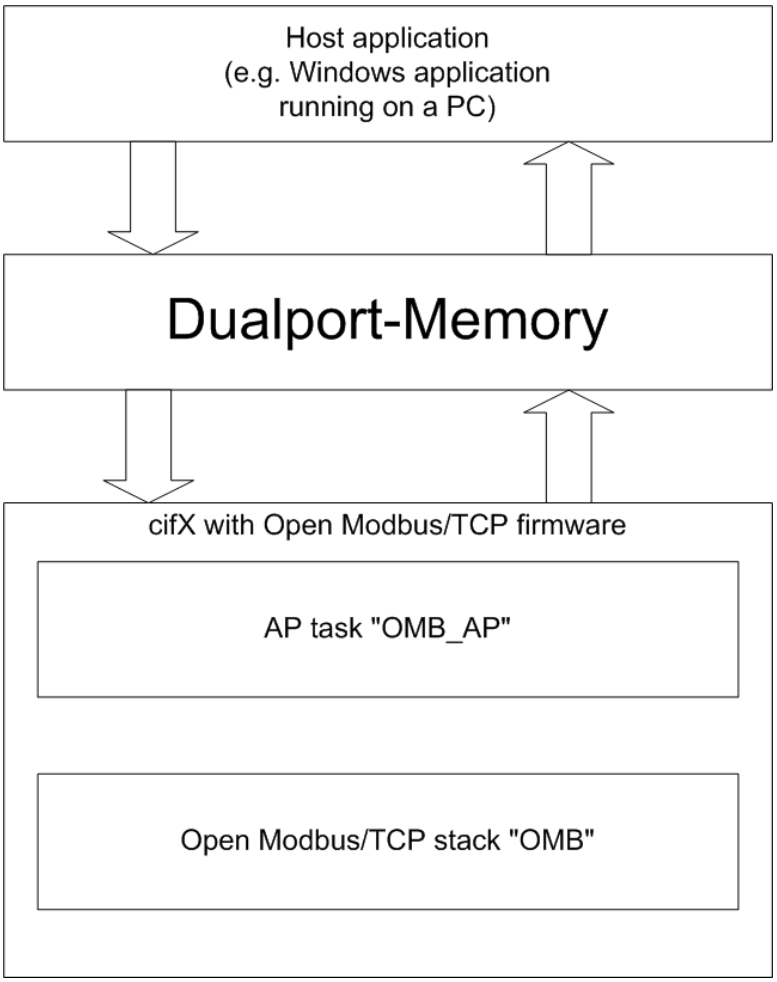


Figure 3: Host application accesses the Open Modbus/TCP

The host application communicates with the OMB_AP task via the dual-port memory by exchanging packets. The following chapter describes the packets that may be received or sent by the OMB_AP Task. In general packets can be divided into “Request/Confirmation” packets and Indication/Response packets. “Request” packets are sent from the host application to the Open Modbus/TCP stack and the stack will return a corresponding “Confirmation”. Also the Open Modbus/TCP stack can send “Indication” packets to the host application and the host application has to send a corresponding “Response” packet.

The OMB_AP task internally communicates with the under laid OMB Task and the CMD Task and TCP/IP Task.

4.1 Configuration

4.1.1 Configuration via packet

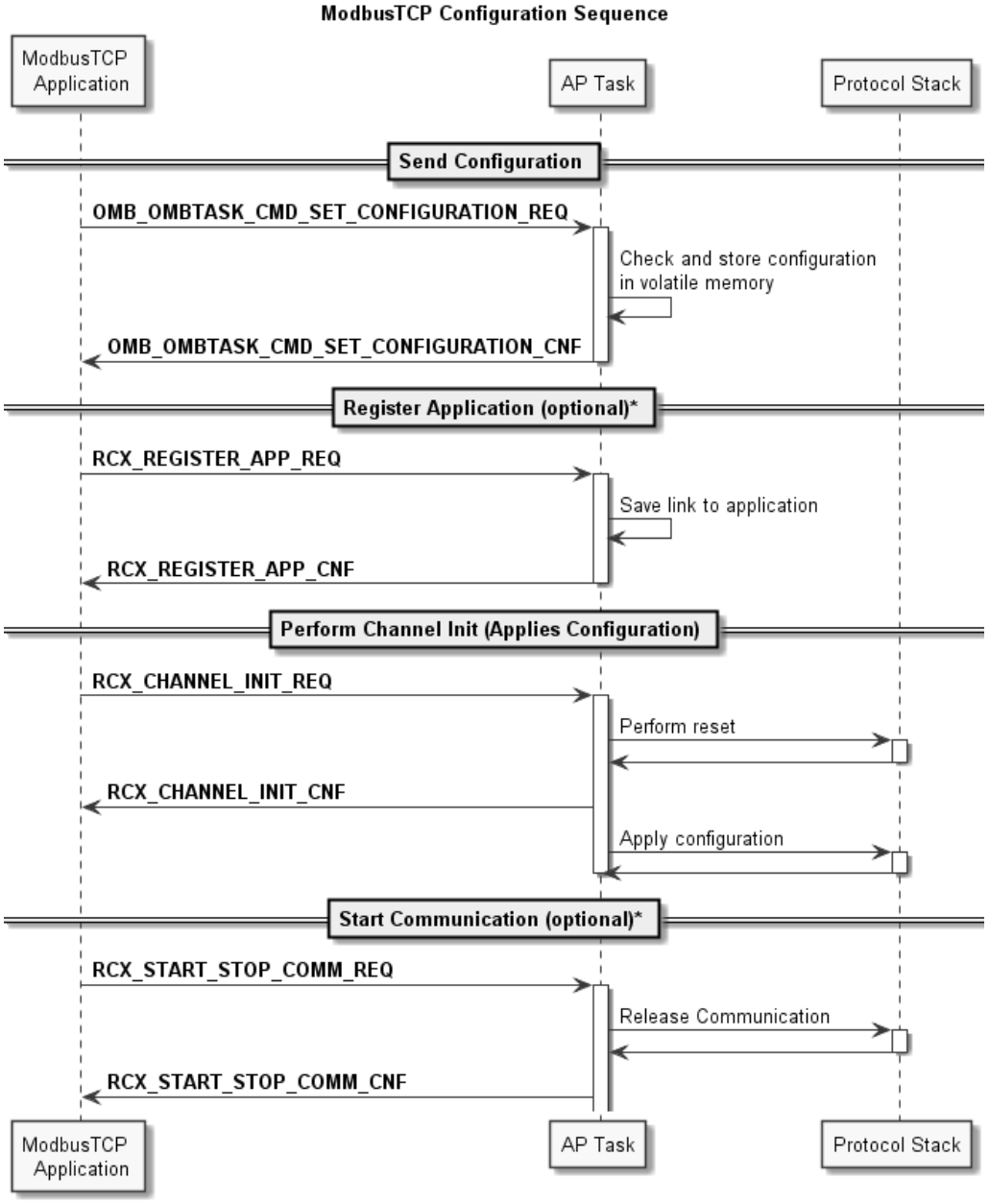


Figure 4: Configuration sequence

*) These packets are optional.

The diagram above shows the Open Modbus/TCP configuration flow via packet API.

In order to configure the Open Modbus/TCP stack, following packet configuration procedure is required:

Packet	Explanation
OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ/CNF – Set configuration (page 34)	This packet is required to provide the configuration data. The configuration data are verified. In case of failure no data are accepted and a corresponding error code is set in the confirmation packet. In case of success the configuration parameters are stored internally in RAM.
RCX_REGISTER_APP_REQ_REQ/CNF – Register application	This packet is optional. It is not required in 'IO-Mode'. It is required in *message mode* in order to receive indications packets. For more information, see reference [1].
RCX_CHANNEL_INIT_REQ/CNF – Channel init and configuration activation	Perform a channel initialization to activate the configuration. For more information, see reference [1].
RCX_START_STOP_COMM_REQ/CNF – Start/stop communication on the Bus	Start the Communication This packet is optional. It is only required if the startup behavior has been set 'Application controlled' mode. For more information, see reference [1].

Table 19: Configuration sequence

4.1.2 Configuration via config file inibatch.nxd

Additionally it is possible to configure the Open Modbus/TCP stack with a data base file called ‘inbatch.nxd’.

This configuration file ‘inibatch.nxd’ is created by the configuration software ‘SyCon.net’ or the ‘netX Configuration Tool’. This configuration file is typically downloaded and stored into flash file system.

The configuration file ‘inbatch.nxd’ contains the following packets:

1. OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ/CNF – Set configuration (page 34)
2. RCX_CHANNEL_INIT_REQ/CNF – Channel init and configuration activation (see reference [1])

During system startup the operating system rcX is opening this file, extract the two packets and send them to OMB AP Task. This means the configuration flow is the same like when user sends the packets from host application.

4.1.3 Configuration of the command table

If the Open Modbus/TCP stack operates as client with command table as describe in chapter Command table (on page 16) it can be configured:

- By a configuration file command.nxd created by SyCon.net
- By packets form the host application

From Open Modbus/TCP V2.6 it is possible to configure the command table fully by the host application without SyCon.net data base. The following figure shows the configuration sequence when the command table is configured by the host application. The command table configuration packets are described in chapter Message mode: Client with command table on page 72.

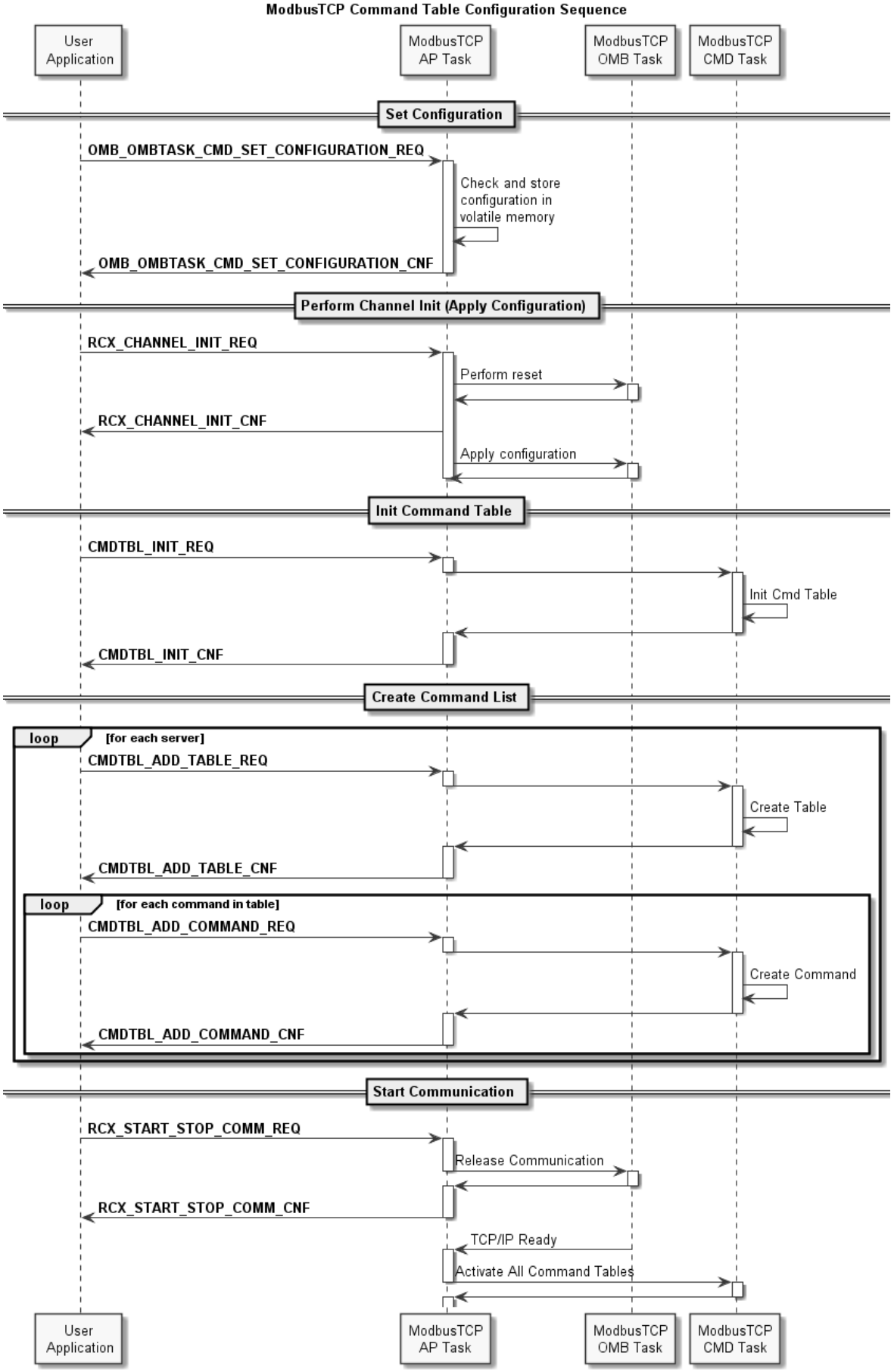


Figure 5: Command table configuration sequence

4.1.4 OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ/CNF – Set configuration

This packet provides the configuration parameters of the Open Modbus/TCP stack.

The following applies:

- Configuration parameters will be stored internally volatile in RAM.
- In case of any error no data will be stored at all.
- A channel init is required to activate the parameterized data.
- This packet does not perform any application registration at the stack automatically.
- Application registration must be done with a separate packet 'RCX_REGISTER_APP_REQ' which is described in the netX Dual-Port-Memory Manual [1].
- Application registration is only required when receiving of indication packets is whished.
- This packet will be denied if the configuration lock flag is set.

Packet structure reference

```
/* Basic Modbus/TCP Configuration - structure*/
typedef struct OMB_OMBTASK_CONFIG_Ttag
{
  TLR_UINT32  ulOpenServerSockets;    /* number of sockets to open  */
  TLR_UINT32  ulAnswerTimeout;        /* Internal Timeout           */
  TLR_UINT32  ulOmbOpenTime;          /* Time to close Socket       */
  TLR_UINT32  ulMode;                 /* Message or IO-Mode         */
  TLR_UINT32  ulSendTimeout;          /* Parameter for TCP-Task     */
  TLR_UINT32  ulConnectTimeout;       /* Parameter for TCP-Task     */
  TLR_UINT32  ulCloseTimeout;         /* Parameter for TCP-Task     */
  TLR_UINT32  ulSwap;                 /* Swap Data or not           */
} OMB_OMBTASK_CONFIG_T;

/* TCP/IP Configuration- structure */
typedef struct TCPIP_DATA_IP_CMD_SET_CONFIG_REQ_Ttag
{
  TLR_UINT32  ulFlags;
  TLR_UINT32  ulIpAddr;
  TLR_UINT32  ulNetMask;
  TLR_UINT32  ulGateway;
  TLR_UINT8   abEthernetAddr[6];
} TCPIP_DATA_IP_CMD_SET_CONFIG_REQ_T;

/* Extended Modbus/TCP Configuration - structure*/
typedef struct OMB_OMBTASK_CONFIG_EXT_Ttag
{
    TLR_UINT32  ulProcessWatchdog;
    TLR_UINT32  ulInactiveTimeout;
    TLR_UINT16  usMaxRegisterNumber;
    TLR_UINT16  usMaxCoilsNumber;
    TLR_UINT16  ausReserved[2];
}OMB_OMBTASK_CONFIG_EXT_T;

/* Ident Configuration – Structure */
typedef struct OMB_OMBTASK_CONFIG_IDENT_Ttag
{
    TLR_CHAR     szVendorName[OMB_OMBTASK_VENDOR_NAME_STR_SIZE];
    TLR_CHAR     szProductCode[OMB_OMBTASK_PRODUCT_CODE_STR_SIZE];
    TLR_CHAR     szRevision[OMB_OMBTASK_MAJ_MIN_REV_STR_SIZE];
    TLR_CHAR     szVendorUrl[OMB_OMBTASK_VENDOR_URL_STR_SIZE];
    TLR_CHAR     szProductName[OMB_OMBTASK_PRODUCT_NAME_STR_SIZE];
    TLR_CHAR     szModelName[OMB_OMBTASK_MODEL_NAME_STR_SIZE];
    TLR_CHAR     szUseAppName[OMB_OMBTASK_USER_APP_NAME_STR_SIZE];
} OMB_OMBTASK_CONFIG_IDENT_T;
```

```
/* Set Configuration Data – Structure */
typedef struct OMB_OMBTASK_DATA_CMD_SET_CONFIGURATION_REQ_Ttag
{
  TLR_UINT32                          ulSystemFlags;   /* common system config flags */
  TLR_UINT32                          ulWdgTime;       /* watchdog time              */
  OMB_OMBTASK_CONFIG_T                tOmbConfig;      /* basic Modbus/TCP config    */
  TCPIP_DATA_IP_CMD_SET_CONFIG_REQ_T tTcpConfig;      /* TCP/IP configuration       */
  TLR_UINT32                          ulFlags;         /* misc configuration flags   */
  /* Parameter starting with V2.6 */
  OMB_OMBTASK_CONFIG_EXT_T            tOmbConfigExt;   /* extend. Modbus/TCP config  */
  OMB_OMBTASK_CONFIG_IDENT_T          tOmbConfigIdent; /* Ident config for FC43      */
} OMB_OMBTASK_DATA_CMD_SET_CONFIGURATION_REQ_T;

/* Set Configuration Packet – Structure */
typedef struct OMB_OMBTASK_PACKET_CMD_SET_CONFIGURATION_REQ_Ttag
{
  TLR_PACKET_HEADER_T                           tHead;
  OMB_OMBTASK_DATA_CMD_SET_CONFIGURATION_REQ_T  tData;
} OMB_OMBTASK_PACKET_CMD_SET_CONFIGURATION_REQ_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_SET_CONFIGURATION_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	482 (62)* (66)**	Packet Data Length in bytes OMB_OMBTASK_DATA_CMD_SET_CONFIGURATION_REQ_SIZE
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Status set 0*
ulCmd	UINT32	0x3F18	OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure OMB_OMBTASK_DATA_CMD_SET_CONFIGURATION_REQ_T			
ulSystemFlags	UINT32	Default value: 0 indicating AUTOSTART Allowed values: 0,1	System Flags BIT 0: AUTOSTART(0) / APPLICATION CONTROLLED(1) BIT 2 - 31 not yet implemented BIT 0 corresponds to OMB_OMBTASK_SYS_FLAG_COM_CONTROLLED_RELEASE The meaning of BIT 0 is: 0 - Communication with a remote station after a device start is allowed without BUS_ON flag, but the communication will be interrupted if the BUS_ON flag changes state to 0 ; 1 - Communication with a remote station is allowed only with the BUS_ON flag.
ulWdgTime	UINT32	0, 20 … 65535	Watchdog time (in ms) 0 means watchdog timer has been switched off.
tOmbConfig	STRUCT		Open Modbus/TCP Basic configuration data structure: OMB_OMBTASK_CONFIG_T See section *Basic parameter* on page 38.
tTcpConfig	STRUCT		TCP/IP configuration data structure: TCPIP_DATA_IP_CMD_SET_CONFIG_REQ_T See section *TCP/IP parameter* on page 39.
ulFlags	UINT32	Bit field	Configuration Flags See section *Parameter flags* on page 41.
Configuration Data starting with Open Modbus/TCP V2.6			
tOmbConfigExt	STRUCT		Open Modbus/TCP Extended configuration data structure: OMB_OMBTASK_CONFIG_EXT_T See section *Extended parameter* on page 42.
tOmbConfigIdent	STRUCT		Modbus TCP Ident configuration data structure: OMB_OMBTASK_CONFIG_IDENT_Ttag See chapter *Ident parameter* on page 43.

Table 20: OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ

* The packet length of 62 bytes is also accepted for compatibility reason with the older firmware version V2.0.x. In this case, the parameter `ulFlags, tOmbConfigExt, tOmbConfigIdent` is set internally to zero.

** The packet length of 66 bytes is also accepted for compatibility reason with the older firmware version V2.5.x. In this case, the parameter `tOmbConfigExt, tOmbConfigIdent` is set internally to zero.

Parameter ulSystemFlags

This parameter contains the system flags area. Currently this value may only have the values 0 or 1. The start of the device can be performed either application controlled or automatically:

- Automatic (0):

  Network connections are opened automatically without taking care of the state of the host application. Communication with a remote controller after a device start is allowed without `BUS_ON` flag, but the communication will be interrupted if the `BUS_ON` flag changes state to 0

- Application controlled (1):

  The Open Modbus/TCP stack is forced to wait for the host application before releasing the network communication. Network communication is allowed with the `BUS_ON` flag set. Setting the `BUS_ON` flag can be performed by using the packet `RCX_START_STOP_COMM_REQ/CNF` packet.

The default value is 0 (Automatic).

Parameter ulWdgTime

This parameter contains the time interval for the supervision of data transfer by the internal watchdog timer. The value must be either the value 0 or a number between 20 and 65535.

If the value 0 is specified, this indicates the watchdog timer has been switched off. Otherwise, the watchdog timer interval is specified in units of milliseconds

4.1.4.1 Basic parameter

Variable	Type	Value / Range	Description
ulOpenServerSockets	UINT32	0..**4**..16	Number of sockets to provide for server requests. A value of 0 would mean that the Open Modbus/TCP task exclusively works as a client while a value of 16 means that the Open Modbus/TCP task exclusively works as server in message mode. The values 1 … 15 means that the Open Modbus/TCP task could work as a client and server simultaneous. Default value: 4
ulAnswerTimeout	UINT32	1...60000	Telegram timeout This parameter is only relevant for client jobs in message mode. After expiration of this time, the job will be canceled and an error is send to the application. The value is multiplied with 100 ms. Default value: 20 (equivalent to 2 seconds effectively) **Note:** This timeout starts after a command is send to the destination device via TCP.
ulOmbOpenTime	UINT32	1...60000	Connection remain open time This parameter sets the connection remain open time. This parameter is only for client jobs in message mode. The connection to the destination device stays open, until timeout is expired. The value is multiplied with 100 ms. Default value: 10 (= 1 s) **Note:** This timeout starts after receiving the answer to a command.
ulMode	UINT32	0, 1	Operation mode of the stack 0: Message mode 1: IO mode (Default)
ulSendTimeout	UINT32	0… 2000000000	Send timeout parameter for TCP-Task in ms Specifies the timeout value for trying to send messages via TCP/IP. Default value: 31000 0: 0 activates the default value 0 represents the default value of 31000 milliseconds. **Note**: This parameter is only for client jobs in message mode.
ulConnectTimeout	UINT32	0… 2000000000	Connect timeout parameter for TCP task in ms Specifies the timeout value for trying to establish a connection with the TCP task. Default value: 31000 0: 0 activates the default value **Note**: This parameter is only for client jobs in message mode.
ulCloseTimeout	UINT32	0… 2000000000	Close timeout parameter for TCP task in ms Specifies the timeout value for trying to close a connection with the TCP task. Default value: 13000 0: 0 activates the default value **Note**: This parameter is only for client jobs in message mode.

Variable	Type	Value / Range	Description
ulSwap	UINT32	0, 1	Data storage mode 0: Data will not be swapped 1: Data will be swapped (default) This parameter decides whether register data will be swapped between application context and network transportation or not. The swap option is valid for both directions: sending and receiving. This means if the swap option is enabled the register values from application context will be swapped before they are send to the network all received register values from the network will be swapped before submitted to the application context. The default value for Swap is “enabled” because register values of Open Modbus/TCP are specified in MSB format. But in most cases the applications are in LSB format.

Table 21: Basic parameter: Structure OMB_OMBTASK_CONFIG_T

The parameters ulSendTimeout, ulConnectTimeout and ulCloseTimeout decide about the timeout between the Open Modbus Task and the TCP Task.

4.1.4.2 TCP/IP parameter

Additionally also some parameters related to the underlying TCP/IP protocol layers need to be configured.

Variable	Type	Value / Range	Description
ulFlags	UINT32	0..63	Bit mask, see below
ulIpAddr	UINT32	Valid IP address	IP address This parameter is only effective, if the IP_CFG_FLAG_IP_ADDR flag is set in ulFlags.
ulNetMask	UINT32	Valid netmask	Netmask This parameter is only effective, if the IP_CFG_FLAG_NET_MASK flag is set in ulFlags.
ulGateway	UINT32	Valid IP address	IP address of gateway This parameter is only effective, if the IP_CFG_FLAG_GATEWAY flag is set in ulFlags.
abEthernetAddr[6]	UINT8[6]	Valid Ethernet address	Ethernet address of the device (6 byte) The abEthernetAddr area can be used to overwrite the device’s default Ethernet address (MAC address). This parameter is only effective, if the IP_CFG_FLAG_ETHERNET_ADDR flag is set in ulFlags.

Table 22: TCP/IP Parameter

Parameter ulFlags

The ulFlags parameter contains bit-oriented flags according to the following table:

Bits	Name (Bit mask)	Description
0	IP_CFG_FLAG_IP_ADDR	IP address available: If set, the ulIpAddr parameter will be evaluated.
1	IP_CFG_FLAG_NET_MASK	Netmask available: If set, the ulNetMask parameter will be evaluated. If the flag is not set the stack will assume to be an isolated host which is not connected to any network. The ulGateway parameter will be ignored in this case.
2	IP_CFG_FLAG_GATEWAY	Gateway available: If set, the ulGateway parameter will be evaluated. If the flag is not set the stack will assume that there exists no gateway.
3	IP_CFG_FLAG_BOOTP	Enable BOOTP: If set, the stack tries to obtain its configuration from a BOOTP server.
4	IP_CFG_FLAG_DHCP	Enable DHCP: If set, the stack tries to obtain its configuration from a DHCP server.
5	IP_CFG_FLAG_ETHERNET_ADDR	Set Ethernet address (MAC address): If set, the abEthernetAddr area will be evaluated.
31 ... 6	Reserved	Reserved for future use

Table 23: Parameter ulFlags

Please note, that a fallback procedure between the different configuration methods is active, if more than one choice is enabled in the ulFlags parameter. If enabled, the stack will first try to configure using DHCP. If DHCP configuration fails, the stack will fall back to BOOTP, if this is enabled. In case of a BOOTP failure, the values found in the ulIpAddr, ulNetMask and ulGateway parameters will be used, if enabled in ulFlags. If none of these configuration mechanisms succeed, the stack will report an error.

4.1.4.3 Parameter flags

This parameter is an Open Modbus/TCP stack related parameter. It holds bit-oriented flags according to the following table. The default value of this flag field is all bit are 0.

Bits	Name (Bit mask)	Description
0	OMB_OMBTASK_CFG_FLAG_FC1_FC3_OUTPUT	Alternative mapping in IO mode If set, the Function codes FC1 and FC3 are mapped to the Output Data image of dual-port memory (abPd0Output[]).Needed e.g. for Clients without FC2, FC4 support. See also chapter *Reading and writing data*.
1	OMB_OMBTASK_CFG_FLAG_TCPIP_NO_CONFIG	Skip Configuration of TCP/IP stack 0 The TCP/IP stack is configure by the Open Modbus/TCP stack 1 The TCP/IP stack is configure is skipped by the Open Modbus/TCP stack If set, the TCP/IP stack is **not** configured. Use this flag only for special cases! This means, in case that someone else configure the TCP/IP stack!
2 … 3 *)	OMB_OMBTASK_CFG_FLAG_FAILSAFE_MASK OMB_OMBTASK_CFG_FLAG_FAILSAFE_SET OMB_OMBTASK_CFG_FLAG_FAILSAFE_HLS	Failsafe behavior: 00b Failsafe mode clear (default) 01b Failsafe mode set 10b Failsafe mode hold last state Only relevant in 'IO_MODE'
4 *)	OMB_OMBTASK_CFG_FLAG_PRESET_INPUTS	If this bit is set the entire input image preset to 0xFF on Set Configuration request. Only relevant in 'IO_MODE'
5 *)	OMB_OMBTASK_CFG_FLAG_SERVER_CONNECTION_IND_FORWARDI NG	Forwarding of Connection Indication packet 0: Disable 1: Enable If this bit is set the packet Connection Indication will be forwarded to the host application
6 … 7	Reserved	Reserved set to 0
8 *)	OMB_OMBTASK_CFG_FLAG_READ_WD_TRIGGER	Read Write Watchdog triggering: 0: Only Write function codes will trigger the ProcessData watchdog (Default) 1: Read and Write function codes will trigger the ProcessData watchdog Only relevant in 'IO_MODE'
9 *)	OMB_OMBTASK_CFG_FLAG_FC7_ENABLE_FORWARDING	FC 7 Forwarding: 0: Forwarding FC 7 disabled (Default) 1: Forwarding FC 7 enabled If this bit is set function code FC 7 is forwarded to host application. Normally, FC 7 is handled stack internally. Only relevant in 'MESSAGE_MODE'
11 ... 31	Reserved	Reserved for future use

Table 24: Parameter ulFlags

Note: The *) marked Flags are introduced with Open Modbus/TCP Stack V2.6

4.1.4.4 Extended parameter

Variable	Type	Value / Range	Description
ulProcessWatchdog	UINT32	**0**, 20 … 65535 (ms)	The process data watchdog monitors read write request from a connected client. This watchdog is only applicable in server IO_MODE. The watchdog is triggered by receiving any read/ write requests from a client. When the watchdog expires the register and coil data which are transferred into DPM input image will be set to the configured fail safe values. 0 (default): The watchdog is disabled.
ulInactiveTimeout	UINT32	**0**, 20 **30000**… 65535 (ms)	The inactivity timeout is a timer that monitors the TCP/IP traffic on an open TCP/IP connection. If no TCP/IP traffic appears in this time the connection will be closed by the stack. The default value is 30000 ms. 0: If the value is set 0 the stack will internally use the default value of 30000 ms.
usMaxRegisterNumber	UINT16	IO mode: 0 … 2880 Message mode: ignored	With the maximum number of register it is possible to limit the allowed accessible Modbus register. The default value is 2880 register this corresponds the max. Size of the dual-port input image of 5760 bytes. If a client tries to access a register above this configured value the Open Modbus/TCP stack will immediately return an exception to the client. Note: This parameter is only effective in IO mode. In message mode it will be ignored.
usMaxCoilsNumber	UINT16	IO mode: 0 … 46080 Message mode: ignored	With the max. Number of coils it is possible to limit the allowed accessible Modbus coils. The default value is 46080 coils this corresponds the max. Size of the dual-port input image of 5760 bytes. If a client tries to access a register above this configured value the Open Modbus/TCP stack will immediately return an exception to the client. Note: This parameter is only effective in IO mode. In message mode it will be ignored.
ausReserved	UINT16[2]	0	Reserved set to 0.

Table 25: Extended Parameter – Structure

Note: The Extended Parameter are introduced with Open Modbus/TCP Stack V2.6

4.1.4.5 Ident parameter

The ident parameters are used internally by the Open Modbus/TCP stack to respond to a FC43 request.

Variable	Type	Value / Range	Description
szVendorName[]	UINT8[64]	zero terminated visible string	The VendorName is a visible zero terminated string which represents the vendor name of a device. e.g. “Hilscher Gesellschaft fuer Systemautomation mbH“
szProductCode[]	UINT8[64]	zero terminated visible string	The ProductCode is a visible zero terminated string which represents a product code of a device. e.g. “ModbusTCP Device”
szRevision[]	UINT8[16]	zero terminated visible string	The Revision is a visible zero terminated string which represents a revision of a device. e.g. “V2.6.0.0”
szVendorUrl[]	UINT8[64]	zero terminated visible string	The VendorUrl is a visible zero terminated string which represents e.g. a web address of the vendor. e.g. “www.hilscher.com”
szProductName[]	UINT8[64]	zero terminated visible string	Product Name e.g. “IO Device”
szModelName[]	UINT8[64]	zero terminated visible string	Model name The ModelName is a visible zero terminated string which represents a name of a device. e.g. “ModbusTCP Device”
szUseAppName[]	UINT8[64]	zero terminated visible string	Application name The UseAppName is a visible zero terminated string which represents an application name. e.g. “ModbusTCP Demo”

Table 26: Open Modbus/TCP Ident configuration data

Note: The Ident parameter are introduced with Open Modbus/TCP Stack V2.6

4.1.5 Confirmation

Packet structure reference

```
typedef struct OMB_OMBTASK_PACKET_CMD_AP_SET_CONFIGURATION_CNF_Ttag
{
  TLR_PACKET_HEADER_T                    tHead;
} OMB_OMBTASK_PACKET_CMD_AP_SET_CONFIGURATION_CNF_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_AP_SET_CONFIGURATION_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32		Destination End Point Identifier, unchanged
ulSrcId	UINT32		Source End Point Identifier, unchanged
ulLen	UINT32	0	Packet Data Length in bytes (OMB_OMBTASK_DATA_CMD_AP_SET_CONFIGURATION_CNF_SIZE)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	0 ... 2^{32}-1	*See section Status/Error codes OMB_AP* task **)*
ulCmd	UINT32	0x3F19	OMB_OMBTASK_CMD_SET_CONFIGURATION_CNF - Command
ulExt	UINT32	0	Extension, untouched
ulRout	UINT32	x	Routing not in use – do not touch

Table 27: OMB_OMBTASK_CMD_SET_CONFIGURATION_CNF –Confirmation of Provide Warmstart Parameters Packet

Note: The status codes beginning with `TLR_E_IP_ERR…` or `TLR_E_TCP_TASK…` originate from the TCP/IP protocol stack. For more information please refer to the TCP/IP manual (Revision 6).

4.2 Common RCX packets

The OMB_AP task is responsible for handling common stack commands. Common stack commands are defined in the command range of the operating system rcX and described in the API Manual of the Hilscher dual-port memory [1]. The common stack commands are implemented in all Hilscher stacks. The reason for the common stack commands is to unify the handling with all Hilscher stacks.

The Open Modbus/TCP stack support following common stack commands listed below.

Common packet	Command
RCX_CHANNEL_INIT_REQ/CNF – Channel init and configuration activation This packet causes a channel initialization and activation of the previously provided configuration data. This packet is an essential packet in conjunction with configuration process of the Open Modbus/TCP stack. Refer to the section Configuration how to use this packet.	0x2F80, 0x2F81
RCX_REGISTER_APP_REQ_REQ/CNF – Register application This packet is used to register the application context to the Open Modbus/TCP stack. It is required in order to receive 'Indication' packets. In 'Message Mode' it is vital to send this packet in order to receive e.g. '`OMB_OMBTASK_CMD_RECEIVE_IND`' packets. In 'IO-Server' mode the registration is not required. **Note**: When sending this packet twice the second registration will be rejected with an error code.	0x2F10, 0x2F11
RCX_UNREGISTER_APP_REQ_ REQ/CNF – Unregister application This packet is used to unregister the application context from the Open Modbus/TCP stack. When the application is unregistered no further 'Indication' packets will be sent from the Open Modbus/TCP stack to the application context.	0x2F12, 0x2F13
RCX_START_STOP_COMM_REQ/CNF – Start/stop communication on the Bus This packet is used to start or stop the network communication. This packet **must** be send after sending 'OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ/CNF – Set configuration' service when the Parameter ulSystemFlags is configure in 'Application controlled' startup behavior. It is **not** required when the startup behavior is configured in 'Automatic' startup. **Note:** When sending this packet with parameter to stop the network communication the Open Modbus/TCP stack will reject all incoming requests from a client directly with an exception code.	0x2F30, 0x2F31
RCX_LOCK_UNLOCK_CONFIG_REQ/CNF – Lock/unlock configuration on the bus This packet is used to enable or disable the so call 'config lock' mechanism within the Open Modbus/TCP stack. This can be used as protection mechanism against unwanted stack operation. **Note:** When the 'config lock' is enable other services e.g. for configuration, start/stop communication or reset are reject with an error code 'TLR_E_CONFIG_LOCK'. RCX_DELETE_CONFIG_REQ/CNF When the 'config lock' is enable other services e.g. for configuration, start/stop communication or reset are reject with an error code 'TLR_E_CONFIG_LOCK'. RCX_DELETE_CONFIG_REQ/CNF	0x2F32, 0x2F33
RCX_DELETE_CONFIG_REQ/CNF	0x2F14, 0x2F15
RCX_SET_WATCHDOG_TIME_REQ/CNF This packet can be used be to set individually the application watchdog 'Parameter ulWdgTime'. This value is the same that has been set with service 'OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ/CNF – Set configuration'.	0x2F04, 0x2F05
RCX_GET_WATCHDOG_TIME_REQ/CNF This packet can be used to read back the current configured application watchdog value described in 'Parameter ulWdgTime'. This value is the same that has been set with service 'OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ/CNF – Set configuration'.	0x2F02, 0x2F03

Common packet	Command
RCX_GET_DPM_IO_INFO_REQ/CNF This packet is used to obtain information about the currently used number of input and output bytes in the dual-port memory. The number of inputs and outputs depends on the configuration. This information can be used by the host application for an optimized IO exchange to copy only the amount of configured IO bytes.	0x2F0C, 0x2F0D
RCX_FIRMWARE_IDENTIFY_REQ/CNF This packet is used to obtain the version of the Open Modbus/TCP stack.	0x1EB6, 0x1EB7
DIAG_INFO_GET_COMMON_STATE_REQ This packet is used to read out the common diagnostic information.	0x2F00, 0x2F01

Table 28: Common rcX packets

4.3 IO mode: Server Modbus data model

The stack handles 5760 bytes for input data and 5760 bytes for output data in the dual-port memory. To exchange data between the device and the application, the application program can use the device driver to read and write the input and output memory. The input and output data images can be found at:

Input and Output Data Images			
Offset	**Type**	**Name**	**Description**
0x1000	UINT8	`abPd0Output[5760]`	Output Data Image Data To The Network
0x2680	UINT8	`abPd0Input[5760]`	Input Data Image Data From The Network

Table 29: Input and Output Data Images

The registers and the coils are located in the **same** memory. Thus, either a word manner or bit manner access to actually the **same** data can be selected!

Reading and writing data

Open Modbus/TCP devices can write and read back data in the input data image `abPd0Input[5760]` and read data from the output data image `abPd0Output[5760]` via the TCP/IP network.

The host applications can write data in the output data image `abPd0Output[5760]` and read data from the input data image `abPd0Input[5760]` via the dual-port memory.

Figure 6 shows the default mapping which means that function code 1, 3 and 23 read data from the input area.

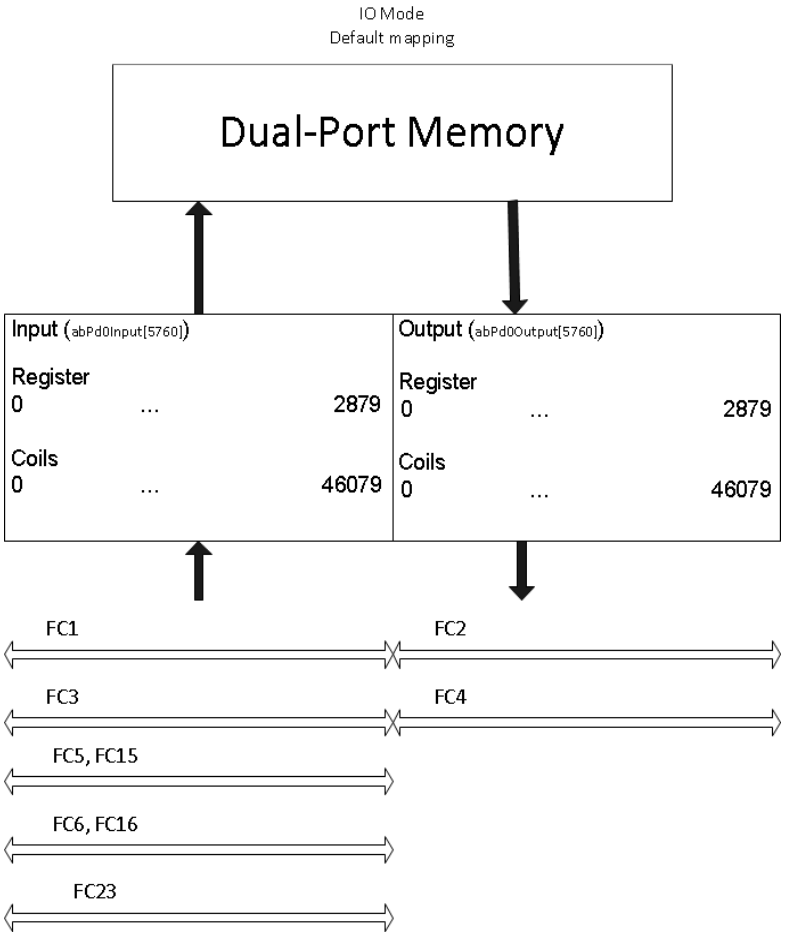


Figure 6: Addressing model of Open Modbus/TCP in IO mode (default mapping)

If an Open Modbus/TCP device writes data to one register with function code 6 or function code 16, another Open Modbus/TCP device is able to read back these data accordingly by using function codes 1 or 3.

Figure 7 shows the alternative mapping which means that function code 1, 3 and 23 read data from the output area. This mapping is needed for clients that do not support function code 2 and function code 4 and allows these clients to use function code 1, 3 or 23 to read data from the output area.

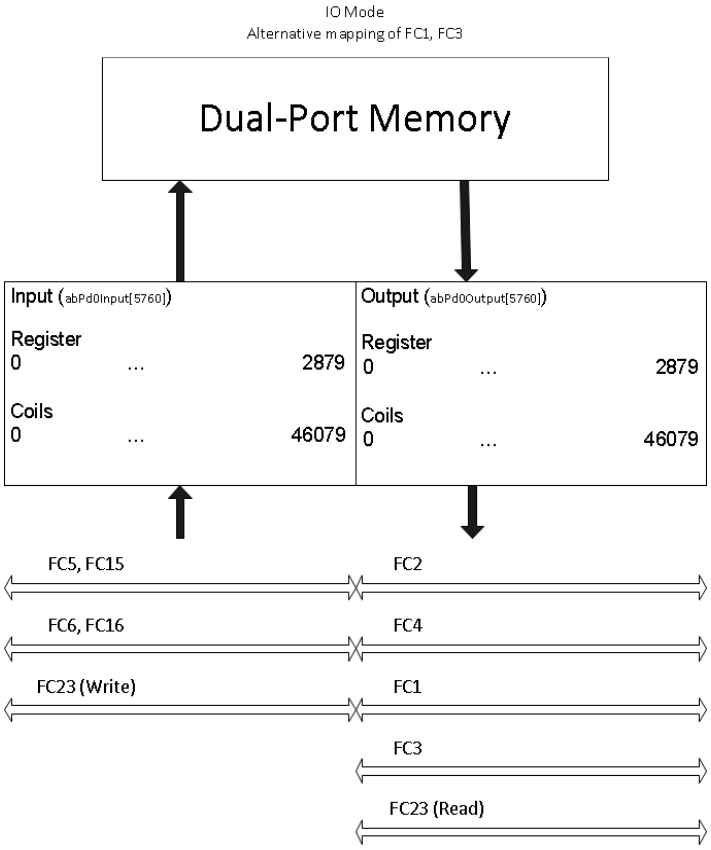


Figure 7: Addressing model of Open Modbus/TCP in IO mode (alternative mapping)

To activate the alternative mapping, flag `OMB_OMBTASK_CFG_FLAG_FC1_FC3_OUTPUT` of parameter `ulFlags` has to be set.

4.4 Message mode: Server with packets

4.4.1 Overview

In Message mode the Open Modbus/TCP stack can run as Modbus/TCP server. In this case all incoming Modbus/TCP requests (function codes) from a remote client are forwarded to the host application. All requests are forwarded with the packet OMB_OMBTASK_CMD_RECEIVE_IND/RES – Receive data indication. This mode is the preferred way to implement a fully user specific Modbus Data model. It is in scope of the user application to decide which function codes are supported. Also it is in scope of the user application which register and coils are supported. The entire Modbus address range is available in this mode.

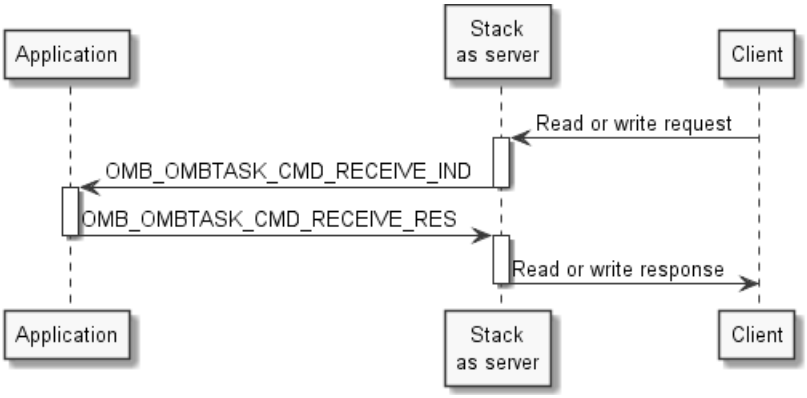


Figure 8: Sequence diagram OMB_OMBTASK_CMD_RECEIVE_IND

To operate in this mode two preconditions need to be full filled

- The Open Modbus/TCP stack must be configured in Message mode (`ulMode = 0`)
- The Open Modbus/TCP stack must have configured some server ports (e.g. `'ulOpenServerSockets' = 4`)
- The host application must be registered with the packet RCX_REGISTER_APP_REQ_REQ/CNF in order to receive the indication packet.

4.4.2 OMB_OMBTASK_CMD_RECEIVE_IND/RES – Receive data indication

This indication occurs on reception of a request of a remote client. Thus the host application has to act as a server by serving these requests with response packets `OMB_OMBTASK_CMD_RECEIVE_RES`. The action to take on reception of `OMB_OMBTASK_CMD_RECEIVE_IND` depends on the function code transmitted along with the indication.

The variables of the request and the confirmation packet have the following meaning:

- Variable `ulRouting` contains the IP address of the client from which the request has been received or to which the response needs to be sent.
- Variable `ulUnitId` contains the unit identifier, i.e. the identification of a remote slave connected on a serial line or on other buses. This variable is initialized by the client. Do not change this value for the response.

- The variable `ulFunctionCode` contains the function code that has been requested by the communication client. Possible function codes FC1, FC2, FC3, FC4, FC5, FC6, FC15, FC16, FC23
- The variable `ulException` contains zero for the indication. The host application can generate an exception in the response packet. The exact behavior of exception generation is the following:
  - If the host application sets `ulSta` unequal to `TLR_S_OK`, the stack generates the exception GATEWAY TARGET DEVICE FAILED TO RESPOND (0x0B)
  - If the host application sends a wrong response (e.g. a wrong packet length `ulLen`), the stack generates also the exception GATEWAY TARGET DEVICE FAILED TO RESPOND (0x0B)
  - If the host application sets `ulSta` equal to `TLR_S_OK` and `ulFunctionCode` greater than 0x7F (means, the host application adds 0x80 to the function code to generate an exception), the stack generates the exception from variable `ulException`.
- Union variable `unData` contains various detail information.

  The contents of `unData` is

```
union
{
  struct
  {
    TLR_UINT32  ulDataAdr;      /* Starting address         */
    TLR_UINT32  ulDataCnt;      /* Register- or Bit-Count   */

    TLR_UINT8   abData[OMB_MAX_DATA_CNT];
  } tFcStd;   /* Union for FCs 1-6, 15-16 */

  struct
  {
    TLR_UINT32  ulDataAdrRead;  /* Read Starting address    */
    TLR_UINT32  ulDataCntRead;  /* Quantity to Read         */
    TLR_UINT32  ulDataAdrWrite; /* Write Starting address   */
    TLR_UINT32  ulDataCntWrite; /* Quantity to Write        */
    TLR_UINT8   abData[OMB_MAX_DATA_CNT];
  } tFc23;    /* Union for FC 23          */

} unData;     /* Data part of PDU         */
```

Union tFcStd for function codes 1-6, 15 and 16

- Variable ulDataAdr contains the register- or bit-offset depending on the requested function code, always beginning at offset zero.
- Variable ulDataCnt contains the register- or bit-count depending on the requested function code.
- The field abData[OMB_MAX_DATA_CNT] contains the user data to be transferred. The field can contain up to 250 bytes of usable data. This is equivalent to:
  - 125 16-bit-registers.
  - 2000 coils

  stored at the same location.

Union tFc23 for function code 23

- Variable ulDataAdrRead contains the read register offset, always beginning at offset zero.
- Variable ulDataCntRead contains the read register count.
- Variable ulDataAdrWrite contains the write register offset, always beginning at offset zero.
- Variable ulDataCntWrite contains the write register count.
- The field abData[OMB_MAX_DATA_CNT] contains the user data to be transferred. The field can contain up to 250 bytes of user data. These are:
  - up to 121 16-bit registers for the write part (indication packet)
  - up to 125 16-bit registers for the read part (response packet).

  For more information on the usage of registers and coils refer to section Modbus function codes on page 16.

Packet structure reference

```
#define OMB_MAX_DATA_CNT    250   /* Maximum user data count in bytes */
                                  /* (125 registers or 2000 coils)    */


typedef  struct OMB_OMBTASK_DATA_CMD_Ttag
{
  TLR_UINT32  ulRouting;          /* IP address               */
  TLR_UINT32  ulUnitId;           /* Unit identifier          */

  TLR_UINT32  ulFunctionCode;     /* Function code (FC)       */
  TLR_UINT32  ulException;        /* Exception code           */

   union
  {
     struct
    {
      TLR_UINT32  ulDataAdr;      /* Starting address         */
      TLR_UINT32  ulDataCnt;      /* Register- or Bit-Count   */

      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFcStd;   /* Union for FCs 1-6, 15-16 */

     struct
    {
      TLR_UINT32  ulDataAdrRead;  /* Read Starting address    */
      TLR_UINT32  ulDataCntRead;  /* Quantity to Read         */
      TLR_UINT32  ulDataAdrWrite; /* Write Starting address   */
      TLR_UINT32  ulDataCntWrite; /* Quantity to Write        */
      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFc23;    /* Union for FC 23          */

  }  unData;     /* Data part of PDU         */

}  OMB_OMBTASK_DATA_CMD_T;


typedef struct OMB_OMBTASK_PACKET_CMD_RECEIVE_IND_Ttag
{
  TLR_PACKET_HEADER_T     tHead;
  OMB_OMBTASK_DATA_CMD_T  tData;

} OMB_OMBTASK_PACKET_CMD_AP_RECEIVE_IND_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_RECEIVE_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Handle from Register AP's ulSrcId (command OMB_OMBTASK_CMD_REGISTER_AP_REQ)
ulSrcId	UINT32	0 … 15	Source End Point Identifier, specifying the origin of the packet inside the Source Process Socket number `ulSocketNumber` of the receiving OMB socket.
ulLen	UINT32	 24+n (FC1 … 6, 15, 16) 32+n (FC23)	Packet Data Length in bytes (header excluded, variable length depending on the transmitted data) OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_ FC_STD + n OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_ FC23 + n n is the Application data count of abData[250] in bytes n = 0 ... OMB_MAX_DATA_CNT (250)* * The maximum value depends on Function code, see also *Table 31: OMB_OMBTASK_CMD_RECEIVE_IND - Packet length for* length calculation
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32		*See section Status/Error codes OMB_AP* task
ulCmd	UINT32	0x3F06	OMB_OMBTASK_CMD_RECEIVE_IND - Command
ulExt	UINT32	0	Extension not in use, do not touch
ulRout	UINT32	x	Routing not in use
Structure OMB_OMBTASK_DATA_CMD_T			
ulRouting	UINT32		IP address of remote station (Modbus client)
ulUnitId	UINT32	0 … 255	Unit identifier
ulFunctionCode	UINT32	1…6, 15, 16, 23	Function code
ulException	UINT32	0	Exception code
unData	union		Contains various data, see above

Table 30: OMB_OMBTASK_CMD_RECEIVE_IND – Receive Data Indication

Packet length ulLen

Function code	Packet length `ulLen`
FC1	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD`
FC2	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD`
FC3	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD`
FC4	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD`
FC5	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD` + `OMB_FC5_PACKET_LEN`
FC6	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD` + `OMB_FC6_PACKET_LEN`
FC15	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD` + `OMB_BYTES_OF_COIL( unData.tFcStd.ulDataCnt )`
FC16	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC_STD` + `OMB_BYTES_OF_REG(unData.tFcStd.ulDataCnt )`
FC23	`OMB_OMBTASK_DATA_CMD_RECEIVE_IND_SIZE_FC23` + `OMB_BYTES_OF_REG( unData.tFc23.ulDataCntWrite )`

Table 31: OMB_OMBTASK_CMD_RECEIVE_IND - Packet length

Packet structure reference

```
#define OMB_MAX_DATA_CNT    250   /* Maximum user data count in bytes */
                                  /* (125 registers or 2000 coils)    */

typedef  struct OMB_OMBTASK_DATA_CMD_Ttag
{
  TLR_UINT32  ulRouting;          /* IP address               */
  TLR_UINT32  ulUnitId;           /* Unit identifier          */

  TLR_UINT32  ulFunctionCode;     /* Function code (FC)       */
  TLR_UINT32  ulException;        /* Exception code           */

  union
  {
     struct
    {
      TLR_UINT32  ulDataAdr;      /* Starting address         */
      TLR_UINT32  ulDataCnt;      /* Register- or Bit-Count   */

      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFcStd;   /* Union for FCs 1-6, 15-16 */

     struct
    {
      TLR_UINT32  ulDataAdrRead;  /* Read Starting address    */
      TLR_UINT32  ulDataCntRead;  /* Quantity to Read         */
      TLR_UINT32  ulDataAdrWrite; /* Write Starting address   */
      TLR_UINT32  ulDataCntWrite; /* Quantity to Write        */
      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFc23;    /* Union for FC 23          */

  }  unData;     /* Data part of PDU         */

}  OMB_OMBTASK_DATA_CMD_T;

typedef struct OMB_OMBTASK_PACKET_CMD_RECEIVE_RES_Ttag
{
  TLR_PACKET_HEADER_T     tHead;
  OMB_OMBTASK_DATA_CMD_T  tData;

} OMB_OMBTASK_PACKET_CMD_AP_RECEIVE_RES_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_RECEIVE_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32		Destination End Point Identifier, unchanged
ulSrcId	UINT32		Source End Point Identifier, unchanged
ulLen	UINT32		Packet Data Length in bytes (header excluded, variable depending on the transmitted data)
		24+n (FC1 … 6, 15, 16)	OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_ FC_STD + n
		32+n (FC23)	OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_ FC23 + n
			n is the Application data count of abData[250] in bytes
			n = 0 ... OMB_MAX_DATA_CNT (250)*
			* The maximum value depends on Function code, see also *Table 33: OMB_OMBTASK_CMD_RECEIVE_RES - Packet length*
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		*See section Status/Error codes OMB_AP* task
ulCmd	UINT32	0x3F07	OMB_OMBTASK_CMD_RECEIVE_RES - Command
ulExt	UINT32	0	Extension, untouched
ulRout	UINT32	x	Routing not in use – do not touch
Structure OMB_OMBTASK_DATA_CMD_T			
ulRouting	UINT32		IP address of remote station (Modbus client), unchanged
ulUnitId	UINT32	0 … 255	Unit identifier, unchanged
ulFunctionCode	UINT32	1…6, 15, 16, 23 (if so, + 0x80)	Function code, unchanged or 0x80 added to generate an exception - see variable ulException in this chapter
ulException	UINT32		Exception code - see variable ulException in this chapter
unData	union		Contains various data, see above

Table 32: OMB_OMBTASK_CMD_RECEIVE_RES– Receive Data Response

Packet length ulLen

Function code	Packet length `ulLen`
FC1	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD` + `OMB_BYTES_OF_COIL( unData.tFcStd.ulDataCnt )`
FC2	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD` + `OMB_BYTES_OF_COIL( unData.tFcStd.ulDataCnt )`
FC3	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD` + `OMB_BYTES_OF_REG(unData.tFcStd.ulDataCnt )`
FC4	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD` + `OMB_BYTES_OF_REG(unData.tFcStd.ulDataCnt )`
FC5	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD` + `OMB_FC5_PACKET_LEN`
FC6	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD` + `OMB_FC6_PACKET_LEN`
FC15	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD`
FC16	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC_STD`
FC23	`OMB_OMBTASK_DATA_CMD_RECEIVE_RES_SIZE_FC23` + `OMB_BYTES_OF_REG( unData.tFc23.ulDataCntRead )`

Table 33: OMB_OMBTASK_CMD_RECEIVE_RES - Packet length

Example

If the remote station `192.168.10.16` (Client) want to write one register to our station (FC 6, data offset = 10, Value = 255), the Open Modbus/TCP stack generates the following indication packet to host application. The host application should poll/receive this packet via the driver function `xSysdeviceGetPacket()`

Variable	Value
`ulDest`	0
`ulSrc`	Queue handle of `OMB` task, does not matter in this context
`ulDestId`	Handle from Register AP's ulSrcId (command `OMB_OMBTASK_CMD_REGISTER_AP_REQ`)
`ulSrcId`	Socket number `ulSocketNumber` of the receiving OMB socket (the allowed range for socket numbers extends from 0 to 15).
`ulLen`	26 (24 + 2)
`ulId`	Is generated automatically
`ulSta`	0
`ulCmd`	0x3F06 (`OMB_OMBTASK_CMD_RECEIVE_IND`)
`ulExt`	Do not change
`ulRout`	Do not change
`ulRouting`	192.168.10.16 (equivalent to `0xC0A80A10`)
`ulUnitId`	Value from client (Node), typically 0
`ulFunctionCode`	6 (Function code for “Write single register”)
`ulException`	0
`ulDataAdr`	10 (“Offset 10”)
`ulDataCnt`	1 (“1 Register”)
`abData[0]`	0 (No swap)
`abData[1]`	255 (No swap)

Table 34: Example: Writing Data via FC6 - Indication

The host application responds to the indication packet via the driver function `xSysdevicePutPacket()` could for instance should look like:

Variable	Value
`ulDest`	Do not change
`ulSrc`	Queue handle of OMB task, does not matter in this context, do not touch
`ulDestId`	Handle from Register AP's ulSrcId (command OMB_OMBTASK_CMD_REGISTER_AP_REQ) , do not touch
`ulSrcId`	Socket number `ulSocketNumber` of the receiving OMB socket. , do not touch
`ulLen`	26 (24 + 2)
`ulSta`	0 (if successful, otherwise TLR_E_FAIL)
`ulCmd`	0x3F07 (OMB_OMBTASK_CMD_RECEIVE_RES)
`ulExt`	Do not change
`ulRout`	Do not change
`ulRouting`	192.168.10.16, do not change
`ulUnitId`	Do not change
`ulFunctionCode`	6 (Function code “Write single register”), do not change in case of error free response
`ulDataAdr`	10 (“Offset 10”), do not change
`ulDataCnt`	1 (“1 Register”) , do not change
`abData[0]`	0 (No swap), do not change
`abData[1]`	255 (No swap), do not change

Table 35: Example: Writing Data via FC6 – Response

4.4.3 OMB_OMBTASK_CMD_CONNECTION_IND/RES – Connection indication

This indication occurs when ever a **server** connection to the stack has been open or closed. This indication can be used by the host application to implement an access regulation. E.g. the first incoming connection is the connection which is allowed to execute read write operations. The second incoming connection may be restricted to read operations only.

Note: To receive the OMB_OMBTASK_CMD_CONNECTION_IND packet the host application must be registered with RCX_REGISTER_APPLICATION_REQ **AND** it must be enable explicitly in the SET_CONFIGURATION_PACKET with flag: ‚OMB_OMBTASK_CFG_FLAG_SERVER_CONNECTION_IND_FORWARDING‘.

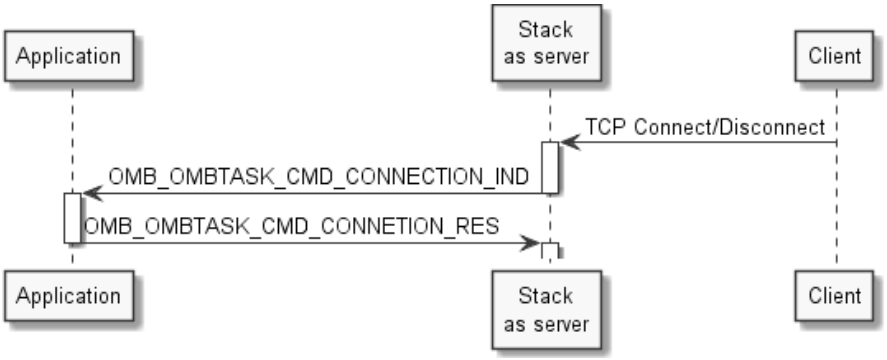


Figure 9: Sequence diagram OMB_OMBTASK_CMD_CONNECTION_IND

Packet structure reference

```
#define OMB_OMBTASK_CONNECTION_STATE_DISCONNECTED (0x00)
#define OMB_OMBTASK_CONNECTION_STATE_CONNECTED    (0x01)
#define OMB_OMBTASK_CONNECTION_TYPE_SERVER        (1)



typedef  struct OMB_OMBTASK_DATA_CONNETCTION_IND_Ttag
{
  TLR_UINT32  ulConnectionId;     /* Connection Identifier    */
  TLR_UINT8   bStatus;            /* Connection status        */
  TLR_UINT8   bType;              /* Connection Type          */
  TLR_UINT16  usReserved;         /* Reserved                 */
  TLR_UINT32  ulIpAddress;        /* Connected IP Address     */

} OMB_OMBTASK_DATA_CONNETCTION_IND_T;


typedef struct OMB_OMBTASK_PACKET_CMD_CONNECTION_IND_Ttag
{
  TLR_PACKET_HEADER_T                tHead;
  OMB_OMBTASK_DATA_CONNECTION_IND_T  tData;

} OMB_OMBTASK_PACKET_CMD_CONNECTION_IND_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_CONNETION_IND_T			Type: Indication
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination Queue-Handle
ulSrc	UINT32		Source Queue-Handle
ulDestId	UINT32		Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Handle from Register AP's ulSrcId (command OMB_OMBTASK_CMD_REGISTER_AP_REQ)
ulSrcId	UINT32	0 … 15	Source End Point Identifier, specifying the origin of the packet inside the Source Process Socket number `ulSocketNumber` of the receiving OMB socket.
ulLen	UINT32	12	Packet Data Length in bytes OMB_OMBTASK_DATA_CMD_CONNECTION_IND_SIZE
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Packet status, zero for indication*
ulCmd	UINT32	0x3F1A	OMB_OMBTASK_CMD_CONNECTION_IND - Command
ulExt	UINT32	0	Extension not in use, do not touch
ulRout	UINT32	x	Routing not in use
Structure OMB_OMBTASK_CONNECTION_IND_DATA_CMD_T			
ulConnectionId	UINT32	0 … 15	Connection Identifier (same as packet ulSrcId)
bState	UINT8	0, 1	Connection State: Bit D0 := 0 -> Not Connected Bit D0 := 1 -> Connected Bit D1 … D7 := Reserved #define OMB_OMBTASK_CONNECTION_STATE_CONNECTED (0x01)
bType	UINT8	1	Connection Type: 1 := Server Connection
usReserved	UINT16	0	Reserved
ulIpAddress	UINT32	n	Client IpAddress that has been connected or disconnected

Table 36: OMB_OMBTASK_CMD_CONNECTION_IND – Connection Indication Packet

Packet structure reference

```
typedef struct OMB_OMBTASK_PACKET_CMD_CONNECTION_RES_Ttag
{
  TLR_PACKET_HEADER_T     tHead;

} OMB_OMBTASK_PACKET_CMD_CONNECTION_RES_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_CONNETION_RES_T			Type: Response
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32		Destination End Point Identifier, unchanged
ulSrcId	UINT32		Source End Point Identifier, unchanged
ulLen	UINT32	0	Packet Data Length in bytes (header excluded, variable depending on the transmitted data)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	0	*See section 6.2 Status/Error codes OMB_AP* task
ulCmd	UINT32	0x3F1B	OMB_OMBTASK_CMD_RECEIVE_RES - Command
ulExt	UINT32	0	Extension, untouched
ulRout	UINT32	x	Routing not in use – do not touch

Table 37: OMB_OMBTASK_CMD_CONNECTION_RES – Connection Response Packet

4.5 Message mode: Client with packets

4.5.1 Overview

In Message mode the Open Modbus/TCP stack can run as Modbus/TCP client. In this case the user application is the initiator of Open Modbus/TCP requests to any Modbus/TCP server. The user application has to use the packet OMB_OMBTASK_CMD_SEND_REQ/CNF - Send data request and send it the Open Modbus/TCP stack.

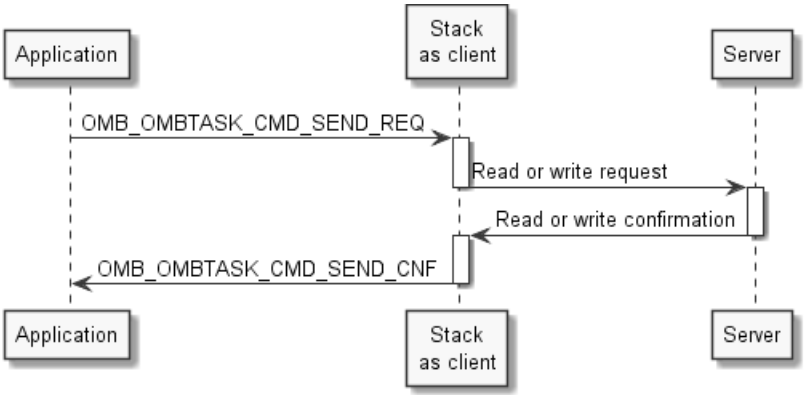


Figure 10: Sequence diagram OMB_OMBTASK_CMD_SEND_REQ

This mode is the preferred way to implement a fully user specific Modbus/TCP Client data model. It is in scope of the user application to decide which function codes are requested to which Modbus/TCP server. The entire Modbus address range is available in this mode.

To operate in this mode two preconditions need to be full filled

- The Open Modbus/TCP stack must be configured in Message mode (`ulMode = 0`)
- The Open Modbus/TCP stack must have configured some client ports (e.g. `'ulOpenServerSockets' = 4; /* 12 are left as client sockets*/`)

4.5.2 OMB_OMBTASK_CMD_SEND_REQ/CNF - Send data request

This packet is used by the host application to send a request to a remote station via the Modbus connection. Thus the host application can act as an Open Modbus/TCP client, the remote station acts as an Open Modbus/TCP server. The Modbus answer from remote station causes a confirmation packet `OMB_OMBTASK_CMD_SEND_CNF`. The action to take on reception of `OMB_OMBTASK_CMD_SEND_CNF` depends on the function code transmitted along with the request.

The variables of the request and the confirmation packet have the following meaning:

- Variable `ulRouting` contains the IP-Address of Open Modbus/TCP device/application to which the request has to send or from which the response comes from, accordingly.
- Variable `ulUnitId` contains the unit identifier, i.e. the identification of a remote slave connected on a serial line or on other buses. This variable is initialized by the client.
- The variable `ulFunctionCode` contains the function code that the request send to the communication partner at the other end of the Open Modbus/TCP connection:

  The following function codes defined by the Open Modbus/TCP specification are supported by the send data request: FC1, FC2, FC3, FC4, FC5, FC6, FC7, FC15, FC16, FC23

Other values might be allowed in the specification but will cause an error in ulSta of the confirmation packet.

For more information about the function codes and their meaning and use see section Modbus function codes on page 16.

- The variable `ulException` contains zero for the request packet. The remote station (server) can generate an exception in the Modbus response. In this case, the confirmation packet is:
  - ulLen = OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD (FC1 … FC7, FC15, FC16) / OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC23 (FC23)
  - ulSta = TLR_E_OMB_OMBTASK_ERR_MODBUS
  - ulFunctionCode is unchanged, means the one from request packet (no 0x80 added!)
  - ulException = Exception code from remote station
- Union variable `unData` contains various detail information.

  The contents of `unData` is

```
union
{
  struct
  {
    TLR_UINT32  ulDataAdr;      /* Starting address         */
    TLR_UINT32  ulDataCnt;      /* Register- or Bit-Count   */

    TLR_UINT8   abData[OMB_MAX_DATA_CNT];
  } tFcStd;   /* Union for FCs 1-6, 15-16 */

  struct
  {
    TLR_UINT32  ulDataAdrRead;  /* Read Starting address    */
    TLR_UINT32  ulDataCntRead;  /* Quantity to Read         */
    TLR_UINT32  ulDataAdrWrite; /* Write Starting address   */
    TLR_UINT32  ulDataCntWrite; /* Quantity to Write        */
    TLR_UINT8   abData[OMB_MAX_DATA_CNT];
  } tFc23;    /* Union for FC 23          */

} unData;     /* Data part of PDU         */
```

Union `tFcStd` for function codes 1-7, 15 and 16

- Variable `ulDataAdr` contains the register- or bit-offset depending on the function code, always beginning at offset zero.
- Variable `ulDataCnt` contains the register- or bit-count depending on the function code.
- The field `abData[OMB_MAX_DATA_CNT]` contains the user data to be transferred. The field can contain up to 250 bytes of usable data. This is equivalent to:
  - 125 16-bit-registers.
  - 2000 coils

  stored at the same location.

Union `tFc23` for function code 23

- Variable `ulDataAdrRead` contains the read register offset, always beginning at offset zero.
- Variable `ulDataCntRead` contains the read register count.
- Variable `ulDataAdrWrite` contains the write register offset, always beginning at offset zero.
- Variable `ulDataCntWrite` contains the write register count.
- The field `abData[OMB_MAX_DATA_CNT]` contains the user data to be transferred. The field can contain up to 250 bytes of usable data. These are:
  - up to 121 16-bit-registers for the write part (request packet)
  - up to 125 16-bit-registers for the read part (confirmation packet).

  For more information on the usage of registers and coils refer to section Modbus function codes on page 16.

Packet structure reference

```
#define OMB_MAX_DATA_CNT    250   /* Maximum user data count in bytes */
                                  /* (125 registers or 2000 coils)    */


typedef  struct OMB_OMBTASK_DATA_CMD_Ttag
{
  TLR_UINT32  ulRouting;          /* IP address               */
  TLR_UINT32  ulUnitId;           /* Unit identifier          */

  TLR_UINT32  ulFunctionCode;     /* Function code (FC)       */
  TLR_UINT32  ulException;        /* Exception code           */

   union
  {
     struct
    {
      TLR_UINT32  ulDataAdr;      /* Starting address         */
      TLR_UINT32  ulDataCnt;      /* Register- or Bit-Count   */

      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFcStd;   /* Union for FCs 1-6, 15-16 */

     struct
    {
      TLR_UINT32  ulDataAdrRead;  /* Read Starting address    */
      TLR_UINT32  ulDataCntRead;  /* Quantity to Read         */
      TLR_UINT32  ulDataAdrWrite; /* Write Starting address   */
      TLR_UINT32  ulDataCntWrite; /* Quantity to Write        */
      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFc23;    /* Union for FC 23          */

  }  unData;     /* Data part of PDU         */

}  OMB_OMBTASK_DATA_CMD_T;


typedef struct OMB_OMBTASK_PACKET_CMD_SEND_REQ_Ttag
{
  TLR_PACKET_HEADER_T     tHead;
  OMB_OMBTASK_DATA_CMD_T  tData;

} OMB_OMBTASK_PACKET_CMD_SEND_REQ_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_SEND_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20/QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	24+n (FC1 … 7, 15, 16) 32+n (FC23)	Packet Data Length in bytes (header excluded, variable length depending on the transmitted data) OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD+ n OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC23 + n n is the Application data count of abData[250] in bytes n = 0 ... OMB_MAX_DATA_CNT (250)* * The maximum value depends on Function code, see also *Table 39: OMB_OMBTASK_CMD_SEND_REQ - Packet length* for length calculation
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	Set to 0
ulCmd	UINT32	0x03F08	OMB_OMBTASK_CMD_SEND_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure OMB_OMBTASK_DATA_CMD_T			
ulRouting	UINT32		IP address of remote station (Modbus server)
ulUnitId	UINT32	0 … 255	Unit identifier
ulFunctionCode	UINT32	1…7, 15, 16, 23	Function code
ulException	UINT32	0	Exception code
unData	union		Contains various data, see above

Table 38: OMB_OMBTASK_CMD_SEND_REQ - Send Data Request

Packet length ulLen

Function code	Packet length `ulLen`
FC1	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD`
FC2	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD`
FC3	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD`
FC4	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD`
FC5	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD` + `OMB_FC5_PACKET_LEN`
FC6	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD` + `OMB_FC6_PACKET_LEN`
FC7	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD`
FC15	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD` + `OMB_BYTES_OF_COIL( unData.tFcStd.ulDataCnt )`
FC16	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC_STD` + `OMB_BYTES_OF_REG(unData.tFcStd.ulDataCnt )`
FC23	`OMB_OMBTASK_DATA_CMD_SEND_REQ_SIZE_FC23` + `OMB_BYTES_OF_REG( unData.tFc23.ulDataCntWrite )`

Table 39: OMB_OMBTASK_CMD_SEND_REQ - Packet length

Packet structure reference

```
#define OMB_MAX_DATA_CNT    250   /* Maximum user data count in bytes */
                                  /* (125 registers or 2000 coils)    */


typedef  struct OMB_OMBTASK_DATA_CMD_Ttag
{
  TLR_UINT32  ulRouting;          /* IP address               */
  TLR_UINT32  ulUnitId;           /* Unit identifier          */

  TLR_UINT32  ulFunctionCode;     /* Function code (FC)       */
  TLR_UINT32  ulException;        /* Exception code           */

   union
  {
     struct
    {
      TLR_UINT32  ulDataAdr;      /* Starting address         */
      TLR_UINT32  ulDataCnt;      /* Register- or Bit-Count   */

      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFcStd;   /* Union for FCs 1-6, 15-16 */

     struct
    {
      TLR_UINT32  ulDataAdrRead;  /* Read Starting address    */
      TLR_UINT32  ulDataCntRead;  /* Quantity to Read         */
      TLR_UINT32  ulDataAdrWrite; /* Write Starting address   */
      TLR_UINT32  ulDataCntWrite; /* Quantity to Write        */
      TLR_UINT8   abData[OMB_MAX_DATA_CNT];
    }  tFc23;    /* Union for FC 23          */

  }  unData;     /* Data part of PDU         */

}  OMB_OMBTASK_DATA_CMD_T;


typedef struct OMB_OMBTASK_PACKET_CMD_SEND_CNF_Ttag
{
  TLR_PACKET_HEADER_T     tHead;
  OMB_OMBTASK_DATA_CMD_T  tData;

} OMB_OMBTASK_PACKET_CMD_SEND_CNF_T;
```

Packet description

Structure OMB_OMBTASK_PACKET_CMD_SEND_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32		Destination queue handle, unchanged
ulSrc	UINT32		Source queue handle, unchanged
ulDestId	UINT32		Destination End Point Identifier, unchanged
ulSrcId	UINT32		Source End Point Identifier, unchanged
ulLen	UINT32	24+n (FC1 … 7, 15, 16) 32+n (FC23)	Packet Data Length in bytes (header excluded, variable depending on the transmitted data) OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD+ n OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC23+ n n is the Application data count of abData[250] in bytes n = 0 ... OMB_MAX_DATA_CNT (250)* * The maximum value depends on Function code, see also *Table 41: OMB_OMBTASK_CMD_SEND_CNF - Packet length*
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32		*See section* Status/Error codes
ulCmd	UINT32	0x3F09	OMB_OMBTASK_CMD_SEND_CNF- Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	x	Routing not in use – do not touch
Structure OMB_OMBTASK_DATA_CMD_T			
ulRouting	UINT32		IP address of remote station (Modbus server), unchanged
ulUnitId	UINT32	0 … 247	Unit identifier, unchanged
ulFunctionCode	UINT32	1…7, 15, 16, 23	Function code, unchanged - see variable ulException in this chapter
ulException	UINT32		Exception code - see variable ulException in this chapter
unData	union		Contains various data, see above

Table 40: OMB_OMBTASK_CMD_SEND_CNF - Send Data Confirmation

Packet length ulLen

Function code	Packet length `ulLen`
FC1	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD` + `OMB_BYTES_OF_COIL( unData.tFcStd.ulDataCnt )`
FC2	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD` + `OMB_BYTES_OF_COIL( unData.tFcStd.ulDataCnt )`
FC3	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD` + `OMB_BYTES_OF_REG(unData.tFcStd.ulDataCnt )`
FC4	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD` + `OMB_BYTES_OF_REG(unData.tFcStd.ulDataCnt )`
FC5	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD`
FC6	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD`
FC7	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD + 1`
FC15	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD`
FC16	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC_STD`
FC23	`OMB_OMBTASK_DATA_CMD_SEND_CNF_SIZE_FC23` + `OMB_BYTES_OF_REG( unData.tFc23.ulDataCntRead )`

Table 41: OMB_OMBTASK_CMD_SEND_CNF - Packet length

Example

If the host application wants to read one holding register from remote station `192.168.10.16` (FC 3, data offset = 10, Value = 255), the host application must send the following request packet to the Open Modbus/TCP stack.

Variable	Value
`ulDest`	0x20 (`RCX_PACKET_DEST_DEFAULT_CHANNEL`)
`ulSrc`	0
`ulDestId`	0
`ulSrcId`	0 (Source End Point Identifier - could be used from host application as handle or so)
`ulLen`	24 (`OMB_OMBTASK_DATA_CMD_SEND_REQ_ SIZE_FC_STD`)
`ulId`	0 (Packet identification - could be used from host application)
`ulSta`	0
`ulCmd`	0x 3F08 (`OMB_OMBTASK_CMD_SEND_REQ`)
`ulExt`	0 (Set to zero)
`ulRout`	0
`ulRouting`	192.168.10.16 (equivalent to 0xC0A80A10)
`ulUnitId`	0
`ulFunctionCode`	3 (Function code for *“Read holding register”*)
`ulException`	0
`ulDataAdr`	10 (“Offset 10”)
`ulDataCnt`	1 (“1 Register”)

Table 42: Example: Reading data via FC3 - Request

The confirmation of the remote station generates a confirmation packet to the host application. The host application should poll/receive this packet via the driver function.

Variable	Value
ulDest	0x20 (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	0
ulDestId	0
ulSrcId	0 (Source End Point Identifier - value from request packet)
ulLen	26 (OMB_OMBTASK_DATA_CMD_SEND_CNF_ SIZE_FC_STD + 2)
ulId	0 (Packet identification - value from request packet)
ulSta	0 (Error free case)
ulCmd	0x 3F09 (OMB_OMBTASK_CMD_SEND_CNF)
ulExt	
ulRout	
ulRouting	192.168.10.16 (equivalent to 0xC0A80A10)
ulUnitId	0
ulFunctionCode	3 (Function code for “Read holding register”)
ulException	0
ulDataAdr	10 (“Offset 10”)
ulDataCnt	1 (“1 Register”)
abData[0]	0 (No swap)
abData[1]	255 (No swap)

Table 43: Example: Reading data via FC3 - Confirmation

4.6 Message mode: Client with command table

4.6.1 Overview

The CMD task is responsible for execution the command table. The command table is list of configured Open Modbus/TCP function codes which are executed cyclically. When the command table is configured the Modbus/TCP stack operates as client. Depending on the function code data are read from a server or written to a remote server. The data that are read or written to a server are exchanged via Input and Output area of dual-port memory with the host application.

The command table can be configured by:

- configuration data base created by Sycon.net
- configuration packets from the host application

The packet API of the CMD task is accessible via OMB_AP – Task of the Open Modbus/TCP stack. This means if the host wants to use the configuration packets to configure its own command table the host has to send them to the OMB_AP – Task of the Open Modbus/TCP stack. The AP – Task will route this packets then to the CMD – Task.

Rules for creating a command list:

- At startup the CMDTBL_INIT_REQ must be send
- For each server that should be requested from the command list a separate ‘TABLE’ must be created
- Each command within a one table must be directed to the same server
- The default number of addressable remote server is 12
- The maximum number of addressable remote server is 16. Therefore the parameter ‘`ulOpenServerSockets`’ must be set 0 to allow the maximum available TCP/IP sockets to be used as client connections.
- The maximum number of configurable commands over all tables is 256.

The following figure shows a command list with a set of commands which are directed to 3 servers. Therefore the command list has to be created with 3 tables.

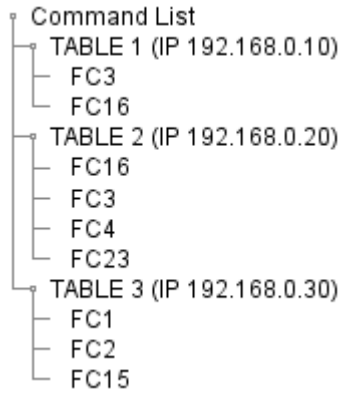


Figure 11: Example command table

The CMD – Task has specified following commands:

Command	Value	Description	Section
CMDTBL_INIT_REQ/CNF	0x0000A300 / 0x0000A301		4.6.2
CMDTBL_ADD_TABLE_REQ/CNF	0x0000A302 / 0x0000A303	Create a new command table	4.6.3
CMDTBL_DELETE_TABLE_REQ/CNF	0x0000A304 / 0x0000A305	Delete a command table	4.6.4
CMDTBL_ACTIVATE_TABLE_REQ/CNF	0x0000A306 / 0x0000A307	Activate a command table	4.6.5
CMDTBL_DEACTIVATE_TABLE_REQ/CNF	0x0000A308 / 0x0000A309	Deactivate a command table	4.6.6
CMDTBL_ADD_COMMAND_REQ/CNF	0x0000A30A / 0x0000A30B	Add a command to a table	4.6.7
CMDTBL_DELETE_COMMAND_REQ/CNF	0x0000A30C / 0x0000A30D	Delete a command from a table	4.6.8
CMDTBL_ACTIVATE_COMMAND_REQ/CNF	0x0000A30E / 0x0000A30F	Activate a command from a table	4.6.9
CMDTBL_DEACTIVATE_COMMAND_REQ/CNF	0x0000A310 / 0x0000A311	Deactivate a command from a table	4.6.10
CMDTBL_TRIGGER_COMMAND_REQ/CNF	0x0000A312 / 0x0000A313	Trigger a command	4.6.11
CMDTBL_GET_IO_INFO_REQ/CNF	0x0000A314 / 0x0000A315	Get IO information	4.6.12
CMDTBL_DEINIT_REQ/CNF	0x0000A316 / 0x0000A317	Deinit	4.6.13
CMDTBL_START_STOP_REQ/CNF	0x0000A318 / 0x0000A319	Table Start/Stop	4.6.14

Table 44: List of CMD task commands

4.6.2 CMDTBL_INIT_REQ/CNF – Init command table task

This command must be send once at startup to initialize and provide some basic configuration to the CMD – Task.

Packet structure reference

```
typedef struct CMDTBL_INIT_DATA_REQ_Ttag
{
  uint32_t ulProtocolType;
  uint32_t ulDpmBitFieldOffset;
} CMDTBL_INIT_DATA_REQ_T;

typedef struct CMDTBL_INIT_REQ_Ttag
{
  TLR_PACKET_HEADER_T       tHead;
  CMDTBL_INIT_DATA_REQ_T    tData;
} CMDTBL_INIT_REQ_T;
```

Packet description

Structure CMDTBL_INIT_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	24	Packet Data Length in bytes sizeof(CMDTBL_INIT_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA300	CMDTBL_INIT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure CMDTBL_INIT_DATA_REQ_T			
ulProtocolType	UINT32	0x12	The command table should run as Open Modbus/TCP command table CMDTBL_PROTOCOL_TYPE_MODBUS_TCP
ulDpmBitFieldOffset	UINT32	0 … 5679	Start offset of the diagnostic bitfield in the dual-port memory input image. The default position should be dword aligned after the last input byte in the input image

Table 45: CMDTBL_INIT_REQ_T packet

Packet structure reference

```
typedef struct CMDTBL_INIT_CNF_Ttag
{
  TLR_PACKET_HEADER_T       tHead;
} CMDTBL_INIT_CNF_T;
```

Packet description

Structure CMDTBL_INIT_REQ_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	x	*See section* Status/Error codes
ulCmd	UINT32	0xA301	CMDTBL_INIT_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	x	Routing not in use – do not touch

Table 46: CMDTBL_INIT_CNF_T packet

4.6.3 CMDTBL_ADD_TABLE_REQ/CNF – Add new command table

Packet structure reference

```
typedef struct CMDTBL_TABLE_Ttag
{
  uint32_t ulProtocolType;
  uint32_t ulCycleTime;
  uint32_t ulInterCommandDelay;
  uint32_t ulInterScanDelay;
  uint32_t ulFlags;
  uint8_t  abReserved[16];
} CMDTBL_TABLE_T;

typedef CMDTBL_TABLE_T CMDTBL_ADD_TABLE_DATA_REQ_T;
```

Packet description

Structure CMDTBL_ADD_TABLE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	36	Packet Data Length in bytes sizeof(CMDTBL_ADD_TABLE_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA302	CMDTBL_ADD_TABLE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure CMDTBL_ADD_TABLE_DATA_REQ_T			
ulProtocolType	UINT32	0x12	The command table should run as an Open Modbus/TCP command table CMDTBL_PROTOCOL_TYPE_MODBUS_TCP
ulCycleTime [ms]	UINT32	**0** .. 60.000	Cycle time in that all commands of this of this shell be executed once. The cycle time is given in ms. Note: If the execution of all commands within the command table takes longer than this time, the cycle time cannot be respected and the cycle time will be increased. If the execution of all commands within a command table takes less than this time, the next execution of the command table will be started when this time has expired. The default value is 0 which means the table is executed is fast as possible. When the last command is finished immediately with the first command will be started again.
ulInterCommand Delay [ms]	UINT32	**0** .. 60.000	This delay time specifies a fix time that is inserted between two commands of command table. It can be used to delay the execution of commands e.g. for a non-performant server that needs a pause on each request. This time is valid for all commands within one table. The default value is 0. This means no delay is inserted between each command.

Structure CMDTBL_ADD_TABLE_REQ_T			Type: Request
ulInterScanDelay t [ms]	UINT32	**0** .. 60.000	This delay time can be used to insert a fix delay time when the last command of a table was executed before start over with the first command of the table. The default value is 0. This means no delay time is inserted.
ulFlags	UINT32	**0…1**	Configuration flag field: Per default all falgs are 0 Bit D0 : Table disable bit If this bit is set the table (all commands within the table) is excluded from the execution when the command table is started. The table can be activated later on explicitly at run time with the CMDTBL_ACTIVATE_TABLE_REQ Request.
abReserved[16]	UINT8[]	0	Reserved array set to 0

Table 47: CMDTBL_ADD_TABLE_REQ_T packet

Packet structure reference

```
typedef struct CMDTBL_ADD_TABLE_DATA_CNF_Ttag
{
  uint32_t ulTableId;
}
CMDTBL_ADD_TABLE_DATA_CNF_T;
```

Packet description

Structure CMDTBL_ADD_TABLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	4	Packet Data Length in bytes sizeof(CMDTBL_ADD_TABLE_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA303	CMDTBL_INIT_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	x	Routing not in use – do not touch
Structure CMDTBL_ADD_TABLE_DATA_CNF_T			
ulTableId	UINT32	n	This is the returned table identifier. The host application has to store this identifier has to use it later as table reference with other commands like add a command or activate a table etc.

Table 48: CMDTBL_ADD_TABLE_CNF_T packet

4.6.4 CMDTBL_DELETE_TABLE_REQ/CNF – Delete a command table

Packet structure reference

```
typedef struct CMDTBL_TABLE_Ttag
{
  uint32_t
} CMDTBL_TABLE_T;

typedef CMDTBL_TABLE_T CMDTBL_ADD_TABLE_DATA_REQ_T;
```

Packet description

Structure CMDTBL_DELETE_TABLE_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes sizeof(CMDTBL_DELETE_TABLE_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA304	CMDTBL_DELETE_TABLE_DATA_REQ_T - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure CMDTBL_DELETE_TABLE_DATA_REQ_T			
ulTableId	UINT32	n	Id of the command table that should be deleted.

Table 49: CMDTBL_DELET_TABLE_REQ_T packet

Packet structure reference

```
typedef struct CMDTBL_TABLE_Ttag
{
  uint32_t ulTableId
} CMDTBL_TABLE_T;

typedef CMDTBL_TABLE_T CMDTBL_DELETE_TABLE_DATA_CNF_T;
```

Packet description

Structure CMDTBL_DELETE_TABLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	2	Packet Data Length in bytes sizeof(CMDTBL_DELET_TABLE_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA305	CMDTBL_DELETE_TABLE_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	x	Routing not in use – do not touch
Structure CMDTBL_DELETE_TABLE_DATA_CNF_T			
ulTableId	UINT32	n	Table identifier of the deleted table.

Table 50: CMDTBL_DELETE_TABLE_CNF_T packet

4.6.5 CMDTBL_ACTIVATE_TABLE_REQ/CNF – Activate a command table

Packet structure reference

```
typedef struct CMDTBL_ACTIVATE_TABLE_DATA_REQ_Ttag
{
  uint32_t ulTableId;   /* Table ID returned by the CMDTBL_ADD_TABLE_CNF_T packet
                           or 0 for all tables created into the database */
}
CMDTBL_ACTIVATE_TABLE_DATA_REQ_T;

typedef CMDTBL_ACTIVATE_TABLE_DATA_REQ_T CMDTBL_ACTIVATE_TABLE_DATA_CNF_T;
```

Packet description

Structure CMDTBL_ACTIVATE_TABLE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes sizeof(CMDTBL_ACTIVATE_TABLE_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA306	CMDTBL_ACTIVATE_TABLE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure CMDTBL_ACTIVATE_TABLE_DATA_REQ_T			
ulTableId	UINT32	n	Command table that should be activated. Use 0 to activate all tables at once that are created. Use table ID returned by the CMDTBL_ADD_TABLE_CNF_T packet to active a particular table.

Table 51: CMDTBL_ACTIVATE_TABLE_REQ_T packet

Packet description

Structure CMDTBL_ACTIVATE_TABLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	2	Packet Data Length in bytes sizeof(CMDTBL_ACTIVATE_TABLE_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA307	CMDTBL_ACTIVATE_TABLE_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_ACTIVATE_TABLE_DATA_CNF_T			
ulTableId	UINT32	n	Table identifier of the activated table.

Table 52: CMDTBL_ACTIVATE_TABLE_CNF_T packet

4.6.6 CMDTBL_DEACTIVATE_TABLE_REQ/CNF – Activate a command table

Packet structure reference

```
typedef struct CMDTBL_DEACTIVATE_TABLE_DATA_REQ_Ttag
{
  uint32_t ulTableId;   /* Table ID returned by the CMDTBL_ADD_TABLE_CNF_T packet
                           or 0 for all tables created into the database */
}
CMDTBL_DEACTIVATE_TABLE_DATA_REQ_T;

typedef CMDTBL_ACTIVATE_TABLE_DATA_REQ_T CMDTBL_DEACTIVATE_TABLE_DATA_CNF_T;
```

Packet description

Structure CMDTBL_DEACTIVATE_TABLE_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes sizeof(CMDTBL_DEACTIVATE_TABLE_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA308	CMDTBL_DEACTIVATE_TABLE_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure CMDTBL_DEACTIVATE_TABLE_DATA_REQ_T			
ulTableId	UINT32	n	Command table that should be deactivated. Use 0 to deactivate all tables at once that are created. Use table ID returned by the CMDTBL_ADD_TABLE_CNF_T packet to deactive a particular table.

Table 53: CMDTBL_DEACTIVATE_TABLE_REQ_T packet

Packet description

Structure CMDTBL_DEACTIVATE_TABLE_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	4	Packet Data Length in bytes sizeof(CMDTBL_DEACTIVATE_TABLE_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA309	CMDTBL_DEACTIVATE_TABLE_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_DEACTIVATE_TABLE_DATA_CNF_T			
ulTableId	UINT32	n	Table identifier of the deactivated table.

Table 54: CMDTBL_DEACTIVATE_TABLE_CNF_T packet

4.6.7 CMDTBL_ADD_COMMAND_REQ/CNF – Add a command to a table

Packet structure reference

```
typedef struct CMDTBL_COMMAND_Ttag
{
  uint32_t ulDeviceAddr;
  uint32_t ulUnitId;
  uint32_t ulFunctionCode;
  uint32_t ulRegisterWriteAddr;
  uint32_t ulRegisterWriteCount;
  uint32_t ulRegisterReadAddr;
  uint32_t ulRegisterReadCount;
  uint32_t ulDPMSrcOffset;
  uint32_t ulDPMDstOffset;
  uint32_t ulTriggerType;
  uint32_t ulCyclePeriod;
  uint32_t ulFlags;
  uint32_t aulReserved[8];
} CMDTBL_COMMAND_T;

typedef struct CMDTBL_ADD_COMMAND_DATA_REQ_Ttag
{
  uint32_t          ulTableId;
  CMDTBL_COMMAND_T  tCommand;
} CMDTBL_ADD_COMMAND_DATA_REQ_T;
```

Packet description

Structure CMDTBL_ADD_COMMAND_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	68	Packet Data Length in bytes sizeof(CMDTBL_ADD_COMMAND_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA30A	CMDTBL_ADD_COMMAND_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure CMDTBL_ADD_COMMAND_DATA_REQ_T			
ulTableId	UINT32	n	Id of the table where to add the command. Use table ID returned by the CMDTBL_ADD_TABLE_CNF_T packet to add a particular command.
Structure CMDTBL_COMMAND_T			
ulDeviceAddr	UINT32	n	IP address of the requested remote server e.g. 0x192.168.10.16 (= 0xC0A80A10)
ulUnitId	UINT32	0 … 255	Sub addressing with the requested remote server.

Structure CMDTBL_ADD_COMMAND_REQ_T			Type: Request
ulFunctionCode	UINT32	1, 2, 3, 4, 5, 6, 15, 16	Requested function code
ulRegisterWriteAddr	UINT32	0 … 65535	Address within the requested server for writing function codes.
ulRegisterWriteCount	UINT32	0 … x	Number of Register or Coils that shall be written to the server. 0 – no data are written
ulRegisterReadAddr	UINT32	0 … 65535	Address within the requested server for reading function codes.
ulRegisterReadCount	UINT32		Number of Register or Coils that shall be read from the server. 0 – no data are read.
ulDPMSrcOffset	UINT32		Byte offset in dual-port memory “Output area” that holds the data which shall be written to the server. Note: The commands must be added in ascending order in relation to their offset. It is not possible to add a second command with a smaller offset then the previous command.
ulDPMDstOffset	UINT32		Byte offset in dual-port memory “Input area” where the data are stored which are read from the server. Note: The commands must be added in ascending order in relation to their offset. It is not possible to add a second command with a smaller offset then the previous command.
ulTriggerType	UINT32	0,1,2,3	Command Execution 0 := CMDTBL_TRIGGER_TYPE_CYCLIC -> cyclicly depending on a configured interval 1 := CMDTBL_TRIGGER_TYPE_ON_CHANGE -> only in case of data change 2: = CMDTBL_TRIGGER_TYPE_ON_CHANGE_NON_ZERO -> only in case of data change to a non-zero value 3 := CMDTBL_TRIGGER_TYPE_ON_REQUEST -> triggered by external event CMDTBL_TRIGGER_COMMAND_REQ
ulCyclePeriod [ms]	UINT32	0 … 60.000	Each command can be configured with an individual cycle time. This time specifies the execution cycle of this command. The default value is 0. This means that the command is executed within the configured cycle time of the command table itself. Note: This cycle time can be configured only when the cycle time ‘ulCycleTime ‘ and the ‘ulInterScanDelay’ of the command table itself is set to 0.
ulFlags	UINT32	0 … 1	Configuration flag field: Per default all falgs are 0 Bit D0 : Command disable bit If this bit is set the command is excluded from the execution when the command table is started. The command can be activated later on explicitly at run time with the CMDTBL_ACTIVATE_COMMAND_REQ Request.
abReserved	UINT8[16]	0	Reserved field set to 0

Table 55: CMDTBL_ADD_COMMAND_REQ_T packet

```
typedef struct CMDTBL_ADD_COMMAND_DATA_CNF_Ttag
{
  uint32_t ulCommandId;
} CMDTBL_ADD_COMMAND_DATA_CNF_T;
```

Packet description

Structure CMDTBL_ADD_COMMAND_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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	2	Packet Data Length in bytes sizeof(CMDTBL_ADD_COMMAND_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA30B	CMDTBL_ADD_COMMAND_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_ADD_COMMAND_DATA_CNF_T			
ulCommandId	UINT32	n	Command identifier of the added command. The host application should hold this handle to use it later by other commands e.g CMDTBL_ACTIVATE_COMMAND_REQ

Table 56: CMDTBL_ADD_COMMAND_CNF_T packet

4.6.8 CMDTBL_DELETE_COMMAND _REQ/CNF – Delete a command

Packet structure reference

```
typedef struct CMDTBL_DELETE_COMMAND_DATA_REQ_Ttag
{
  uint32_t ulCommandId;
} CMDTBL_DELETE_COMMAND_DATA_REQ_T;

typedef CMDTBL_DELETE_COMMAND_DATA_REQ_T CMDTBL_DELETE_COMMAND_DATA_CNF_T;
```

Packet description

Structure CMDTBL_DELETE_COMMAND_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes sizeof(CMDTBL_DELETE_COMMAND_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA30C	CMDTBL_DELETE_COMMAND_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	x	Routing not in use
Structure CMDTBL_DELETE_COMMAND_DATA_REQ_T			
ulCommandId	UINT32	n	Command that shall be deleted. Use the command ID returned by the CMDTBL_ADD_COMMAND _REQ_T packet to delete a particular command.

Table 57: CMDTBL_DELETE_COMMAND_REQ_T packet

Packet description

Structure CMDTBL_DELETE_COMMAND_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	4	Packet Data Length in bytes sizeof(CMDTBL_DEACTIVATE_TABLE_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA30D	CMDTBL_DEACTIVATE_TABLE_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_DELETE_COMMAND_DATA_CNF_T			
ulCommandId	UINT32	n	Command identifier of the deleted comamnd.

Table 58: CMDTBL_DELETE_COMMAND_CNF_T packet

4.6.9 CMDTBL_ACTIVATE_COMMAND_REQ/CNF – Activate a command

Packet structure reference

```
typedef struct CMDTBL_ACTIVATE_COMMAND_DATA_REQ_Ttag
{
  uint32_t ulCommandId;
} CMDTBL_ACTIVATE_COMMAND_DATA_REQ_T;

typedef CMDTBL_ACTIVATE_COMMAND_DATA_REQ_T CMDTBL_ACTIVATE_COMMAND_DATA_CNF_T;
```

Packet description

Structure CMDTBL_ACTIVATE_COMMAND_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes sizeof(CMDTBL_ACTIVATE_COMMAND_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA30E	CMDTBL_ACTIVATE_COMMAND_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	X	Routing not in use
Structure CMDTBL_ACTIVATE_COMMAND_DATA_REQ_T			
ulCommandId	UINT32	n	Command that shall be activated. Use the command ID returned by the CMDTBL_ADD_COMMAND _REQ_T packet to activate a particular command.

Table 59: CMDTBL_ACTIVATE_COMMAND_REQ_T packet

Packet description

Structure CMDTBL_ACTIVATE_COMMAND_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	4	Packet Data Length in bytes sizeof(CMDTBL_ACTIVATE_COMMAND_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA30F	CMDTBL_ACTIVATE_COMMAND_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_ACTIVATE_COMMAND_DATA_CNF_T			
ulCommandId	UINT32	n	Command identifier of the activated command.

Table 60: CMDTBL_ACTIVATE_COMMAND_CNF_T packet

4.6.10 CMDTBL_DEACTIVATE_COMMAND_REQ/CNF – Deactivate a command

Packet structure reference

```
typedef struct CMDTBL_DEACTIVATE_COMMAND_DATA_REQ_Ttag
{
  uint32_t ulCommandId;
} CMDTBL_DEACTIVATE_COMMAND_DATA_REQ_T;

typedef CMDTBL_DEACTIVATE_COMMAND_DATA_REQ_T CMDTBL_DEACTIVATE_COMMAND_DATA_CNF_T;
```

Packet description

Structure CMDTBL_DEACTIVATE_COMMAND_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes sizeof(CMDTBL_DEACTIVATE_COMMAND_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA310	CMDTBL_DEACTIVATE_COMMAND_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	X	Routing not in use
Structure CMDTBL_DEACTIVATE_COMMAND_DATA_REQ_T			
ulCommandId	UINT32	n	Command that shall be activated. Use the command ID returned by the CMDTBL_ADD_COMMAND _REQ_T packet to activate a particular command.

Table 61: CMDTBL_DEACTIVATE_COMMAND_REQ_T packet

Packet description

Structure CMDTBL_DEACTIVATE_COMMAND_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	4	Packet Data Length in bytes sizeof(CMDTBL_DEACTIVATE_COMMAND_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA311	CMDTBL_DEACTIVATE_COMMAND_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_DEACTIVATE_COMMAND_DATA_CNF_T			
ulCommandId	UINT32	n	Command identifier of the activated command.

Table 62: CMDTBL_ACTIVATE_COMMAND_CNF_T packet

4.6.11 CMDTBL_TRIGGER_COMMAND_REQ/CNF – Trigger a command

Packet structure reference

```
typedef struct CMDTBL_TRIGGER_COMMAND_DATA_REQ_Ttag
{
  uint32_t ulCommandId;
} CMDTBL_TRIGGER_COMMAND_DATA_REQ_T;

typedef CMDTBL_TRIGGER _COMMAND_DATA_REQ_T CMDTBL_TRIGGER_COMMAND_DATA_CNF_T;
```

Packet description

Structure CMDTBL_TRIGGER_COMMAND_REQ_T			**Type: Request**
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	4	Packet Data Length in bytes sizeof(CMDTBL_TRIGGER_COMMAND_DATA_REQ_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA312	CMDTBL_TRIGGER_COMMAND_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	X	Routing not in use
Structure CMDTBL_TRIGGER_COMMAND_DATA_REQ_T			
ulCommandId	UINT32	n	Command that shall be triggerd. Use the command ID returned by the CMDTBL_ADD_COMMAND _REQ_T packet to trigger a particular command.

Table 63: CMDTBL_TRIGGER_COMMAND_REQ_T packet

Packet description

Structure CMDTBL_TRIGGER_COMMAND_CNF_T			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	4	Packet Data Length in bytes sizeof(CMDTBL_TRIGGER_COMMAND_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA313	CMDTBL_TRIGGER_COMMAND_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_TRIGGER_COMMAND_DATA_CNF_T			
ulCommandId	UINT32	N	Command identifier of the activated command.

Table 64: CMDTBL_TRIGGER_COMMAND_CNF_T packet

4.6.12 CMDTBL_GET_IO_INFO_REQ/CNF – Get IO Info

Packet description

Structure CMDTBL_GET_IO_INFO_REQ_T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA314	CMDTBL_GET_IO_INFO_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	X	Routing not in use

Table 65: CMDTBL_GET_IO_INFO_REQ_T packet

Packet structure reference

```
typedef struct CMDTBL_GET_IO_INFO_DATA_CNF_Ttag
{
  uint32_t ulInputByteSize;
  uint32_t ulOutputByteSize;
  uint32_t ulDpmBitFieldPosition;
} CMDTBL_GET_IO_INFO_DATA_CNF_T;
```

Packet description

Structure CMDTBL_GET_IO_INFO_CNF_T			**Type: Confirmation**
Variable	**Type**	**Value / Range**	**Description**
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	2	Packet Data Length in bytes sizeof(CMDTBL_GET_IO_INFO_DATA_CNF_T)
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA313	CMDTBL_GET_IO_INFO_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch
Structure CMDTBL_GET_IO_INFO_DATA_CNF_T			
ulInputByteSize	UINT32	0 … 5680	This value defines last byte used “Input area” of the dual-port memory.
ulOutputByteSize	UINT32	0 … 5680	This value defines last byte used “Output area” of the dual-port memory.
ulDpmBitFieldPosition	UINT32	0 … 5680	Start offset of the diagnostic bit field within the dual-port “Input area”. This value relects the offset that has been configured with the CMDTBL_INIT_REQ - Command

Table 66: CMDTBL_GET_IO_INFO_CNF_T packet

4.6.13 CMDTBL_DEINIT_REQ/CNF – Table Deinitialization

The command table de-initialization can be used to reset the command table task. This command sets the CMD Task into the Power ON state.

Packet description

Structure CMDTBL_DEINIT_REQ _T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	0	Packet Data Length in bytes
ulId	UINT32	0 ... 2^{32}-1	Packet Identification as unique number generated by the Source Process of the Packet
ulSta	UINT32	0	*Set to zero*
ulCmd	UINT32	0xA316	CMDTBL_DEINIT_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	X	Routing not in use

Table 67: CMDTBL_DEINIT_REQ packet

Packet description

Structure CMDTBL_DEINIT_CNF			Type: Confirmation
Variable	**Type**	**Value / Range**	**Description**
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	2	0
ulId	UINT32	0 ... 2^{32}-1	Packet Identification, unchanged
ulSta	UINT32	x	*See section* Status/Error codes
ulCmd	UINT32	0xA317	CMDTBL_DEINIT_CNF - Command
ulExt	UINT32	0	Extension, do not touch
ulRout	UINT32	X	Routing not in use – do not touch

Table 68: CMDTBL_DEINIT_CNF packet

4.6.14 CMDTBL_START_STOP_REQ/CNF – Table Start/Stop

This packet can be is used to stop or start the execution of the command table.

Packet structure reference

```
typedef struct CMDTBL_START_STOP_DATA_REQ_Ttag
{
  uint32_t ulStartStopCommand;
}
CMDTBL_START_STOP_DATA_REQ_T;
```

Packet description

Structure CMDTBL_START_STOP_REQ _T			Type: Request
Variable	**Type**	**Value / Range**	**Description**
Structure TLR_PACKET_HEADER_T			
ulDest	UINT32	0x20 QUE_OMBAPTASK	Destination Queue-Handle (RCX_PACKET_DEST_DEFAULT_CHANNEL)
ulSrc	UINT32	0 ... 2^{32}-1	Source Queue-Handle
ulDestId	UINT32	0	Destination End Point Identifier, specifying the final receiver of the packet within the Destination Process. Set to 0 for the Initialization Packet
ulSrcId	UINT32	0 ... 2^{32}-1	Source End Point Identifier, specifying the origin of the packet inside the Source Process
ulLen	UINT32	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	*Set to zero*
ulCmd	UINT32	0xA318	CMDTBL_START_STOP_REQ - Command
ulExt	UINT32	0	Extension not in use, set to zero for compatibility reasons
ulRout	UINT32	X	Routing not in use
Structure CMDTBL_START_STOP_REQ _T			
ulStartStopCommand	UINT32	0, 1	CMDTBL_STOP_COMMAND = 0x00000000 CMDTBL_START_COMMAND = 0x00000001

Table 69: CMDTBL_START_STOP_REQ packet

4.6.15 Command Table diagnostic with Status bit fields

When the Open Modbus/TCP stack is configured in client mode and a “Command Table” is present, the Open Modbus/TCP stack supports the Hilscher standard bit fields “Configured Slaves”, “Active Slaves” and “Diagnostic Slaves”. In this case the Open Modbus/TCP stack acts like a master on the network. Also the Hilscher common diagnostic structure for master stacks “NETX_MASTER_STATUS_T” is filled.

When the Open Modbus/TCP stack starts it will parse the command table and assign for each found IP address one bit in the bit fields. The bit assignment to the IP addresses is done in the same order like they are configured in the command table. The smallest bit is always assigned to the first configured IP address in command table the second bit is assigned to the next configured IP address and so on. If two or more same IP addresses are configured in the command table, then the bit assignment will be done only for the first IP address.

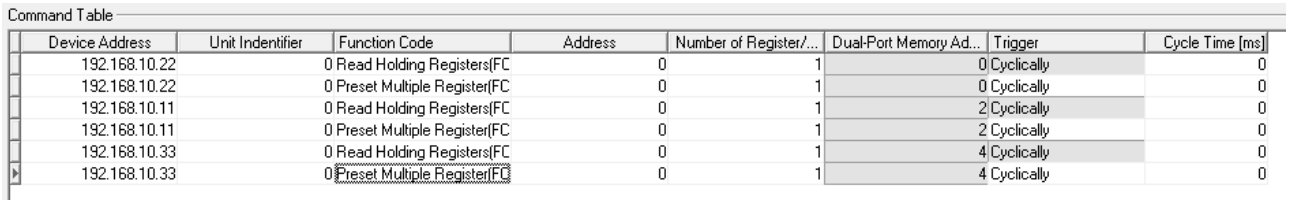
Additionally the OMB Stack supports a bit field per each configured command. The assignment of the command diagnostic flags is in ascending order like they the command sets are defined in the command table.

```
TLR_UINT8   abConfigSlaves[16];  /* Configured Slave bit filed */
TLR_UINT8   abActiveSlaves[16];  /* Active Slave in communication bit field */
TLR_UINT8   abDiagSlaves[16];    /* Slaves with diagnostic bit field*/
TLR_UINT8   abCommandDiag[32];   /* Diag flags per configured command bit field*/
```

The bit fields are transferred within the process data input image of the dual port memory. They are placed 32bit aligned behind the last used input byte of the process data input image. This guarantees the consistency of process data and status information. The bit filed support for configured, active and diagnostic slaves follows the general implementation for all Hilscher master stacks. That’s why the mechanism how to get the exact start offset of each bit field is described generally in the “netX Dual-port Memory Interface” manual. The start offsets and additional information about the bit fields (size, type etc.) can be read out from the “Extended Status Block” in the dual port memory. Refer to chapter “Extended Status Block” in the “netX Dual-port Memory Interface” to get detailed information how to get the bit filed offsets.

Bit Field Example

The figure below shows a command table created with SYCON.net.



Command Table

	Device Address	Unit Indentifier	Function Code	Address	Number of Register/...	Dual-Port Memory Ad...	Trigger	Cycle Time [ms]
	192.168.10.22	0	Read Holding Registers(FC	0	1	0	Cyclically	0
	192.168.10.22	0	Preset Multiple Register(FC	0	1	0	Cyclically	0
	192.168.10.11	0	Read Holding Registers(FC	0	1	2	Cyclically	0
	192.168.10.11	0	Preset Multiple Register(FC	0	1	2	Cyclically	0
	192.168.10.33	0	Read Holding Registers(FC	0	1	4	Cyclically	0
▶	192.168.10.33	0	Preset Multiple Register(FC	0	1	4	Cyclically	0

The command table contains 6 commands. These commands are send to 3 slave different devices with: IPAddress1=192.168.10.22
IPAddress2=192.168.10.11
IPAddress3=192.168.10.33.
The condition is as following IP2 and IP3 are active on the network. IP1 is not connected to the network. The second command (FC16) of IP2 does not work for any reason. All other commands are executed successfully.

In this case, the bit fields are filled as following:

Configured slaves bit field

```
tBitField.abConfigSlaves[0]     = 0b00000111; (bit0 = IP1; bit1 = IP2; bit2 = IP3;)
tBitField.abConfigSlaves[1..15] = 0b00000000;
```

Active slaves bit field

```
tBitField.abActiveSlaves[0]     = 0b00000110; (bit0 = FALSE because IP1 not active)
tBitField.abActiveSlaves[1..15] = 0b00000000;
```

Diagnostic slaves bit field

```
tBitField.abDiagSlaves[0]       = 0b00000010; (bit1 = TRUE -> FC16 of IP2 is faulty)
tBitField.abDiagSlaves[1..15]   = 0b00000000;
```

Command bit field

```
tBitField.abCommandDiag.[0]     = 0b00110100; (bit0;1=0 -> IP1 faulty; bit3=0 -> FC16
                                               of IP2 faulty)
tBitField.abCommandDiag.[1..31] = 0b00000000;
```

4.6.16 Command table timing diagram

The following diagrams show the first typical two cases of the command Table. In this case one cycle time (TCT) is configured for each server command table. This means all commands shall be executed once within this cycle tile. The typical use case is to configure an IO cycle. It is possible to configure an “InterCommandDelay” between each request. This is sometimes required for a slow server. It is also possible to configure a “InterScanDelay”. This will guarantee a break between the last and first command of the table. The fastest execution of the command table can be realized if all timing parameter TCT; ICD and ISD are set to.

The timings are configured with the parameter `ulCycleTime (TCT); ulInterCommandDelay (ICD) and ulInterScanDelay (ISD)` of the command CMDTBL_ADD_TABLE_REQ.

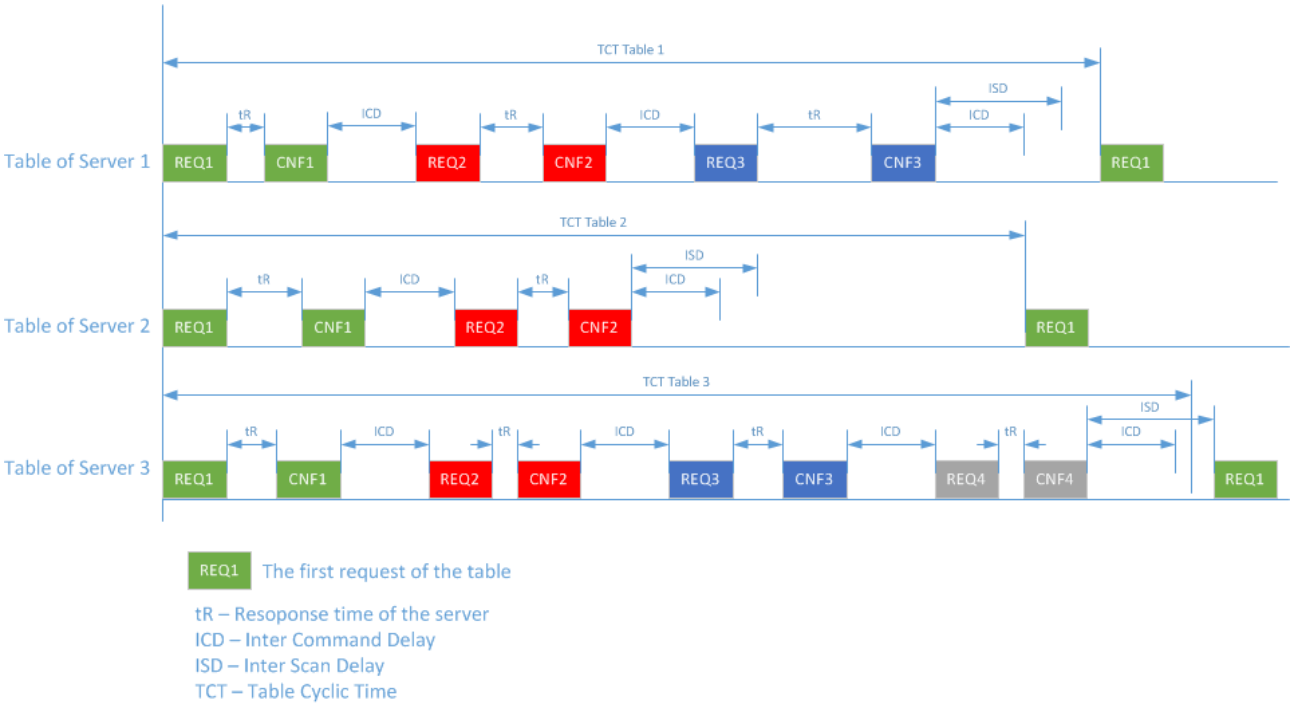


Figure 12: Timing diagram for table cyclic time

The following diagrams show the second typical use cases of the command table. In this case each single command is configure with an individual cycle time (CCP).The typical use case is if some data of a server shall be requested frequently (e.g. IO data) and some status data from the server shall be requested rarely.

The timings are configured with the parameter `ulInterCommandDelay (ICD` of the command CMDTBL_ADD_TABLE_REQ and `ulCyclePeriod` of the command CMDTBL_ADD_COMMAND_REQ.

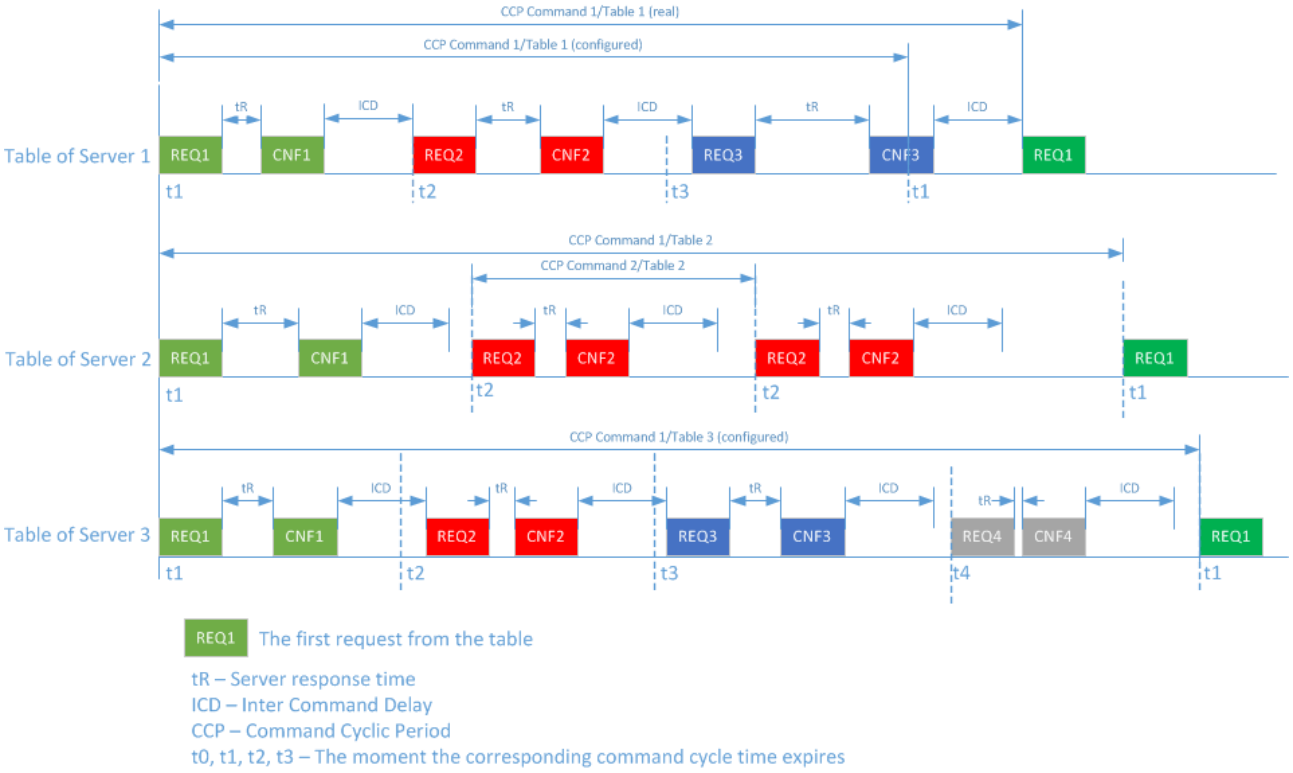


Figure 13: Timing diagram for command cyclic period

Note: It is not possible to configure a mixed mode of both described timing use cases. This means it is not allowed to configure a Table Cycle Time `ulCycleTime (TCT)` to a none 0 value and an individual `ulCyclePeriod` (CCP). If the command individual `ulCyclePeriod` (CCP) is configured then it is not possible to configure a `ulInterScanDelay` (ISD).

5 Special topics

5.1 For programmers

The specified packets in this API manual are defined as C structures in following files:

- OmbOmbTask_Public.h
- CmdTable_Configuration.h
- CmdTable_Configuration.h

Note: These files may contain additional packets and commands which are not described here in this API manual. All non-described packets and commands are for internal use and not for programmer's scope.

5.2 Task start-up parameters

This chapter provides information for users of linkable object modules (LOM).

The task start-up parameters are hand over to the specified tasks which represent the Open Modbus/TCP stack. The task startup parameters are hand over at system startup point when the operating system rcX creates the task. Therefore this section is not relevant for 'loadable firmware' users. In this case the task startup parameters are fixed compiled in into the firmware. They cannot be changed. The task startup parameters are relevant for 'linkable object' users which are writing own software on the netX controller with the operating system rcX.

5.2.1 Start-up parameters of the OMB task

This structure represents a set of the start-up parameter, which can be defined or have to be defined in order to configure the task. The OMB task has the following start-up parameters with their data types, allowed ranges and default values:

Start-up parameters "Version 4" of the OMB task starting with Modbus/TCP V2.6.0

Value	Type	Range	Default	Meaning
`ulQueElemCnt`	UINT32	16 … 16384	256	Process queue size of OMB task
`ulPoolElemCnt`	UINT32	48 ... 2048	64 (equivalent to 4 per socket)	Size of pool elements for indication packets to AP. One pool element allocates (approx.) 1524 bytes

Value	Type	Range	Default	Meaning
ulStartFlags	UINT32	Bit mask	0x00000000	Start flags: 0x000000001 OMB_SRT_FLAG_ENABLE_RANGE When this flag is set the OMB task will evaluate the parameter ulMaxRegsCnt / ulMaxCoilsCnt 0x00000002 OMB_SRT_FLAG_ENABLE_TCP_TASK_SEARCH If this falg is set, the OMB task will search for the TCPIP also on instance 0..6 and not only on the same instance like the OMB task. This flag should be set only in special cases when the TCP task is not placed on same instance like the OMB task. Normally the TCP task should have the same instance like the OMB task. 0x00000004 OMB_SRT_FLAG_ENABLE_TCPIP_NO_CONFIG If this flag is set, the OMB task is forced not to initialize the TCP IP task. e.g. another application is responsible for TCP configuration.
ulOmbCycleEvent	UINT32	10...200	10	Cycle time of OMB task in ms – call interval of cyclic functions. This time must be greater or equal to the OS cycle time **Attention:** A change the default value influences the configured parameter ulAnswerTimeout and ulOmbOpenTime!
ulMaxRegsCnt	UINT32	0 … 2880	0	With this parameter the number of supported register can be limited.
ulMaxCoilsCnt	UINT32	0 … 46080	0	With this parameter the number of supported coils can be limited.
ulNumOfObj	UINT32	0 … n	0	Number of configured FC43 user specific identification data
ptObjList	POINTER	xxx	NULL	Pointer to a stuct array from type ‘OMB_OMBTASK_UESR_SPECIFIC_ID_T’. The struct of this array has ‘ulNumOfObj’ elements.

Table 70: Start-up Parameters of the OMB-Task

5.2.2 Start-up parameters of the OMB_AP task

The OMB_AP-Task has the following start-up parameters with their data types, allowed ranges and default values:

Start-up parameters “Version 4” of the OMB_AP task starting with Modbus/TCP V2.6.0

Value	Type	Range	Default	Meaning
ulQueElemCnt	UINT32	16 … 4096	64	Process queue size of OMB_AP task
ulPoolElemCnt	UINT32	16 ... 2048	32 (equivalent to 2 per socket)	Size of pool elements for indication packets to AP. One pool element allocates (approx.) 1524 bytes
ulStartFlags	UINT32	Bit mask	0x00000000	Start flags 0x00000001 OMBAP_SRT_FLAG_TCP_TASK_SEARCH If this flag is set, the OMB_AP task will search for the TCPIP also on instance 0..6 and not only on the same instance like the OMB_AP task. This flag should be set only in special cases when the TCP task is not placed on same instance like the OMB task. Normally the TCP task should have the same instance like the OMB_AP task.
ulChnInst	UINT32	0 … 3	0	Channel instance of dual-port memory (ulChnInst)
tLedRunGreen	OMB_OMBAPTASK_LED_CONFIG_T	-	-	Configuration of RUN LED (green)
tLedRunRed	OMB_OMBAPTASK_LED_CONFIG_T	-	-	Configuration of RUN LED (red)
tLedErrGreen	OMB_OMBAPTASK_LED_CONFIG_T	-	-	Configuration of ERR LED (green)
tLedErrRed	OMB_OMBAPTASK_LED_CONFIG_T	-	-	Configuration of ERR LED (red)
szCommandTableQueueName	TLR_CHAR[]	-	"QUE_CMD TBL"	Character array that contains the queue name of the CMD-Task. Use the define: #define CMDTBL_QUEUE_NAME
ulCommadTableInstance	UINT32	0 … 3	0	Instance of the CMD-Task queue

Table 71: Start-up Parameters of the OMB_AP-Task

The definition of OMB_OMBAPTASK_LED_CONFIG_T for the LED Configuration is

```
typedef struct OMB_OMBAPTASK_LED_CONFIG_Ttag  OMB_OMBAPTASK_LED_CONFIG_T;

struct OMB_OMBAPTASK_LED_CONFIG_Ttag
{
  STRING      szName[16]; /* Name of LED instance */
  TLR_UINT32  ulLedInst;  /* LED instance         */
};
```

Both LEDs (RUN and ERR) are basically designed as Duo-LEDs. But the Open Modbus/TCP stack serves only the RUN LED (green) and the ERR LED (red). The other two LEDs are switched fix off.

If a LED don’t exist on a hardware (e.g. for a hardware with single LEDs or a hardware without LEDs), set the name of LED instance szName[16] to “” (empty string). Then, this LED instance is not created.

5.2.3 Start-up parameters of the CMD task

The Command Table-Task has the following start-up parameters with their data types, allowed ranges and default values:

Value	Type	Range	Default	Meaning
`ulMaxReadImageSize`	UINT32	0..20000	5680	The maximum size of the input image buffer where the Command Table task should copy the data into by Modbus read commands defined. Corresponds to the size of the DPM Input process data size – the size of diagnostic information mapped into the input image as well. The diagnostic data structure is 80 bytes in size so the input size in this case will be 5760 – 80
`ulMaxWriteImageSize`	UINT32	0..20000	5760	The maximum size of the output image buffer where the Command Table task should copy the data from and send by Modbus write commands defined Corresponds to the size of the DPM Input process data size
`ulWriteImageOffset`	UINT32	0…20000	5760	The offset (in bytes) from the Modbus Register array where the output image should be mapped into Because of compatible reasons with OMB V2.5.x.x it should be located immediately after the end of the input image i.e. 5760
`abDestQueueName[16]`	TLR_CHAR[]	".."	"QUE_OMBTASK"	The queue of the task that should handle the commands to the servers. Normally this is the Modbus Stack Task
`ulDestQueueInst`	UINT32	0	0	The instance of the queue given by the `abDestQueueName parameter`

Table 72: Start-up Parameters of the CMD-Task

6 Status/Error codes

6.1 Status/Error codes OMB task

Hexadecimal value	Definition / description
0x00000000	TLR_S_OK Status ok
0xC0000004	TLR_E_UNKNOWN_COMMAND Unknown Command in Packet received
0xC0000007	TLR_E_INVALID_PACKET_LEN Packet length is invalid.
0xC000001A	TLR_E_REQUEST_RUNNING Request is already running.
0xC0600002	TLR_E_OMB_OMBTASK_SEND_IP_SET_CONFIG_FAILED Failed to forward the SET_CONFIG information to TCP_UDP task (because of a resource problem).
0xC0600003	TLR_E_OMB_OMBTASK_SYSTEM_FUNCTION_CODE Wrong function code.
0xC0600004	TLR_E_OMB_OMBTASK_MOD_MEM_MOD_START_ADR Wrong Modbus start address.
0xC0600005	TLR_E_OMB_OMBTASK_MOD_MEM_LEN IO mode: Wrong length of Memory map.
0xC0600006	TLR_E_OMB_OMBTASK_MOD_MEM_START_MEM_OFF IO mode: Wrong Start byte offset in Memory map.
0xC0600007	TLR_E_OMB_OMBTASK_MOD_MEM_SYSTEM_ERROR IO mode: System error.
0xC0600008	TLR_E_OMB_OMBTASK_INVALID_STARTUP_PARAMETER_QUE_ELEM_CNT Invalid Startup Parameter ulQueElemCnt.
0xC0600009	TLR_E_OMB_OMBTASK_INVALID_STARTUP_PARAMETER_POOL_ELEM_CNT Invalid Startup Parameter ulPoolElemCnt.
0xC060000A	TLR_E_OMB_OMBTASK_INVALID_STARTUP_PARAMETER_START_FLAGS Invalid Startup Parameter ulStartFlags.
0xC060000B	TLR_E_OMB_OMBTASK_INVALID_STARTUP_PARAMETER_OMB_CYCLE_EVENT Invalid Startup Parameter ulOmbCycleEvent.
0xC060000C	TLR_E_OMB_OMBTASK_APPLICATION_TIMER_CREATE_FAILED Failed to create an application timer (Timer task).
0xC060000D	TLR_E_OMB_OMBTASK_APPLICATION_TIMER_INIT_PACKET_FAILED Failed to initialize a packet of application timer (Timer task).
0xC060000E	TLR_E_OMB_OMBTASK_TCP_UDP_IDENTIFY_FAILED Failed to identify the TCP_UDP task.
0xC000000C	TLR_E_WATCHDOG_TIMEOUT Watchdog error occurred.
0xC0000119	TLR_E_NOT_CONFIGURED Configuration not available
0xC060000F	TLR_E_OMB_OMBTASK_TCP_UDP_QUEUE_IDENTIFY_FAILED The queue identification of TCP_UDP task queue has failed.
0xC0600010	TLR_E_OMB_OMBTASK_BUFFER_QUEUE_CREATE_FAILED Creation of buffer queue failed.

Hexadecimal value	Definition / description
0xC0600012	TLR_E_OMB_OMBTASK_FLAGS_VALUE Invalid parameter ‘Flags’ (ulFlags)
0xC0600034	TLR_E_OMB_OMBTASK_SERVER_CONNECT_VALUE Invalid configuration data
0xC0600035	TLR_E_OMB_OMBTASK_ANSWER_TIMEOUT_VALUE Invalid configuration data
0xC0600036	TLR_E_OMB_OMBTASK_OPEN_TIMEOUT_VALUE Invalid parameter 'Omb Open Time' (ulOmbOpenTime).
0xC0600037	TLR_E_OMB_OMBTASK_MODE_VALUE Invalid parameter 'Mode' (ulMode).
0xC0600038	TLR_E_OMB_OMBTASK_SEND_TIMEOUT_VALUE Invalid parameter 'Send Timeout' (ulSendTimeout)
0xC0600039	TLR_E_OMB_OMBTASK_CONNECT_TIMEOUT_VALUE Invalid parameter 'Connect Timeout' (ulConnectTimeout).
0xC060003A	TLR_E_OMB_OMBTASK_CLOSE_TIMEOUT_VALUE Invalid parameter 'Close Timeout' (ulCloseTimeout).
0xC060003B	TLR_E_OMB_OMBTASK_SWAB_VALUE Invalid parameter 'Swap' (ulSwap).
0xC060003C	TLR_E_OMB_OMBTASK_ERR_INIT_TCP_TASK_NOT_READY TCP_UDP task not found
0xC060003D	TLR_E_OMB_OMBTASK_ERR_INIT_PLC_TASK_NOT_READY PLC task not found
0xC0600070	TLR_E_OMB_OMBTASK_ERR_ANSWER TCP_UDP task answered with an error.
0xC0600071	TLR_E_OMB_OMBTASK_ERR_STATE No socket in specific status found.
0xC0600072	TLR_E_OMB_OMBTASK_ERR_VALUE Invalid value in command.
0xC0600073	TLR_E_OMB_OMBTASK_ERR_TCP_TASK_STATE Error in TCP_UDP task state.
0xC0600074	TLR_E_OMB_OMBTASK_ERR_MODBUS Error in Modbus telegram - for further information, see next section.
0xC0600075	TLR_E_OMB_OMBTASK_ERR_NO_SOCKET No free and unused socket found.
0xC0600076)	TLR_E_OMB_OMBTASK_ERR_UNKNOWN_SOCKET TCP_UDP command for an unknown socket received.
0xC0600077	TLR_E_OMB_OMBTASK_ERR_TIMEOUT The timeout for the Client-Job is expired. Timeout-Count starts after target has received the command.
0xC0600078	TLR_E_OMB_OMBTASK_ERR_UNEXPECTED_CLOSE Socket was unexpected closed.
0xC0600079	TLR_E_OMB_OMBTASK_USER_NOT_READY The User is not ready (not registered).
0xC060007A	TLR_E_OMB_OMBTASK_NO_SOCKET_AVAILABLE OMB task is not able to open sockets (TCP_UDP task is not ready).
0xC060007C	TLR_E_OMB_OMBTASK_ERR_IP_CONFIG TCP_UDP task is in configuration status.

Hexadecimal value	Definition / description
0xC060007D	TLR_E_OMB_OMBTASK_PLC_TASK_NOT_INITIALIZED No Dual-port memory access.
0xC060007E	TLR_E_OMB_OMBTASK_SEVER_SOCKET_CLOSED Server Socket is Closed before Answer is received
0xC06000A1	TLR_E_OMB_OMBTASK_DEVICE_ADR Invalid device address (IP address).
0xC06000A5	TLR_E_OMB_OMBTASK_DATA_CNT Invalid Data count.
0xC06000A7	TLR_E_OMB_OMBTASK_FUNCTION Wrong Function code. Function code is not supported.
0xC0600100	TLR_E_OMB_OMBTASK_MOD_DATA_ADR IO mode: Wrong Modbus address. Modbus address is outside of Memory map.
0xC0600101	TLR_E_OMB_OMBTASK_MOD_DATA_CNT IO mode: Wrong Data count in conjunction with the Modbus address. The access area is outside of Memory map.
0xC0600102	TLR_E_OMB_OMBTASK_MOD_FUNCTION_CODE IO mode: Wrong Function code. Function code is not supported.
0xC0600103	TLR_E_OMB_OMBTASK_MOD_DATA_TYPE IO mode: Wrong data type.
0xC0600104	TLR_E_OMB_OMBTASK_MOD_BIT_AREA IO mode: Addressed coil is outside of the IO area.
0xC0600106	TLR_E_OMB_OMBTASK_SEND_TCP_CONFIG_RELOAD_FAILED Failed to forward the configuration reload to TCP_UDP task (because of a resource problem).
0xC0600107	TLR_E_OMB_OMBTASK_WRONG_CONFIG_RELOAD_STS Wrong configuration reload state.
0xC0600108	TLR_E_OMB_OMBTASK_RESOURCE_OCCUPIED System error: The requested resource is occupied.
0xC0600109	TLR_E_OMB_OMBTASK_AP_ALREADY_REGISTERED An application is already registered.
0xC060010A	TLR_E_OMB_OMBTASK_AP_NOT_REGISTERED An application is not registered.
0xC060010B	TLR_E_OMB_OMBTASK_START_STOP_MODE Wrong mode ulMode in command OMB_OMBTASK_CMD_START_STOP_OMB_REQ.
0xC060010C	TLR_E_OMB_OMBTASK_START_STOP_STATE_CHANGE No senseful state change request (Start/stop) in command OMB_OMBTASK_CMD_START_STOP_OMB_REQ.
0xC060010D	TLR_E_OMB_OMBTASK_IO_MODE_COMMAND_INVALID IO mode: Invalid command received
0xC060010E	TLR_E_OMB_OMBTASK_STATE_NOT_RUNNING The OMB stack is not in running state (Info status: ulTaskState is not OMB_ST_TASK_RUNNING) or the Communication state is not operating (ulCommunicationState is not RCX_COMM_STATE_OPERATE).
0xC060010F	TLR_E_OMB_OMBTASK_MBAP_HEADER Wrong MBAP header received (Transaction Identifier, Protocol Identifier)
0xC0600110	TLR_E_OMB_OMBTASK_UNIT_ID Invalid Unit identifier (ulUnitId).
0xC0600111	TLR_E_OMB_OMBTASK_EXCEPTION Invalid Exception code (ulException).

Hexadecimal value	Definition / description
0xC0600112	TLR_E_OMB_OMBTASK_MBAP_LENGTH Invalid MBAP header Length value.
0xC0600113	TLR_E_OMB_OMBTASK_PDU_BYTE_COUNT Invalid PDU Byte count.
0xC0600114	TLR_E_OMB_OMBTASK_PDU_REF_NUMBER Invalid PDU Reference Number (Starting Address).
0xC0600115	TLR_E_OMB_OMBTASK_PDU_DATA_CNT Invalid PDU Data count (Quantity).
0xC0600116	TLR_E_OMB_OMBTASK_PDU_VALUE Invalid PDU Value.
0xC0600117	TLR_E_OMB_OMBTASK_DATA_ADR Wrong Modbus address. The Modbus address is outside of the Modbus Data model (Range 0 ... 65535).
0xC0600118	TLR_E_OMB_OMBTASK_DATA_ADR_CNT Wrong Data count in conjunction with the Modbus address. The access area is outside of the Modbus Data model (Range 0 ... 65535).
0xC0600119	TLR_E_OMB_OMBTASK_INVALID_STARTUP_PARAMETER_OMB_MAX_REGS_CNT Invalid Startup Parameter `ulMaxRegsCnt`.
0xC060011A	TLR_E_OMB_OMBTASK_INVALID_STARTUP_PARAMETER_OMB_MAX_COILS_CNT Invalid Startup Parameter `ulMaxCoilsCnt`.
0xC060011B	TLR_E_OMB_OMBTASK_INVALID_FUNCTION_PARAMETERS Invalid Function Parameter Supplied.
0xC060011C	TLR_E_OMB_OMBTASK_INVALID_FUNCTION_PARAMETERS Any function has been called with invalid parameter.

Table 73: Status/Error codes OMB task

6.2 Status/Error codes OMB_AP task

Hexadecimal value	Definition / description
0x00000000	TLR_S_OK Status ok
0xC0000004	TLR_E_UNKNOWN_COMMAND Unknown Command in Packet received
0xC000001A	TLR_E_REQUEST_RUNNING Request is already running.
0xC0000181	TLR_E_CONFIG_LOCK Changing configuration is not allowed.
0xC0610003	TLR_E_OMB_OMBAPTASK_WATCHDOG_PARAMETER Invalid parameter for watchdog supervision.
0xC0610004	TLR_E_OMB_OMBAPTASK_WATCHDOG_ACTIVATE Failed to activate watchdog supervision.
0xC0610006	TLR_E_OMB_OMBAPTASK_SYS_FLAG_PARAMETER Invalid parameter for system flags
0xC0610007	TLR_E_OMB_OMBAPTASK_ INVALID_STARTUP_PARAMETER_QUE_ELEM_CNT Invalid Startup Parameter ulQueElemCnt.
0xC0610008	TLR_E_OMB_OMBAPTASK_ INVALID_STARTUP_PARAMETER_POOL_ELEM_CNT Invalid Startup Parameter ulPoolElemCnt.
0xC0610009	TLR_E_OMB_OMBAPTASK_ INVALID_STARTUP_PARAMETER_START_FLAGS Invalid Startup Parameter ulStartFlags.
0xC061000A	TLR_E_OMB_OMBAPTASK_ INVALID_STARTUP_PARAMETER_CHN_INST Invalid Startup Parameter ulChnInst.
0xC061000B	TLR_E_OMB_OMBAPTASK_FATAL_ERROR_OMB_TASK The OMB task reports a fatal error. System has stopped. See extended status tMidCodeDiag for further information.
0xC061000C	TLR_E_OMB_OMBAPTASK_COMMAND_EXCEEDS_DPM_SIZE The OMB task reports a fatal error. System has stopped. See extended status tMidCodeDiag for further information.
0xC061000D	TLR_E_OMB_OMBAPTASK_COMMAND_OVERLAPED_DPM_OFFSET The OMB task reports a fatal error. System has stopped. See extended status tMidCodeDiag for further information.
0xC061000E	TLR_E_OMBAPTASK_COMMAND_NOT_ALLOWED_IN_CURRENT_MODE The OMB task reports a fatal error. The packet is not allowed to be sent.
0xC061000F	TLR_E_OMBAPTASK_COMMAND_NOT_ALLOWED_IN_CURRENT_OMB_STATE The packet is not allowed to be sent in case of the Omb stack is in STOP or NOT_CONFGURED state.

Table 74: Status/Error codes OMB_AP task

6.3 Status/Error codes CMD task

Hexadecimal value	Definition / description
0x00000000	TLR_S_OK Status ok
0xC0E20001	TLR_E_CMD_TABLE_COMMAND_INVALID
0xC0E20002L	TLR_E_CMDTBL_INVALID_PROTOCOL_TYPE
0xC0E20003L	TLR_E_CMDTBL_INVALID_QUEUE_NAME
0xC0E20004L	TLR_E_CMDTBL_UNSUPPORTED_VERSION
0xC0E20005L	TLR_E_CMDTBL_TABLE_NOT_AVAILABLE
0xC0E20006L	TLR_E_CMDTBL_COMMAND_NOT_AVAILABLE
0xC0E20007L	TLR_E_CMDTBL_DEST_QUEUE_NOT_INITED
0xC0E20008L	TLR_E_CMDTBL_COMMAND_DOES_NOT_MATCH_
0xC0E20009L	TLR_E_CMDTBL_COMMAND_NOT_TRIGGERABLE
0xC0E2000AL	TLR_E_CMDTBL_COMMAND_NOT_ACTIVE
0xC0E2000BL	TLR_E_CMDTBL_UNSUPPORTED_PROTOCOL_TYPE
0xC0E2000CL	TLR_E_CMDTBL_MAX_TABLES_REACHED
0xC0E2000DL	TLR_E_CMDTBL_MAX_COMMANDS_REACHED
0xC0E2000EL	TLR_E_CMDTBL_ALREADY_INITED
0xC0E2000FL	TLR_E_CMDTBL_NOT_INITED
0xC0E20010L	TLR_E_CMDTBL_REQ_NOT_ALLOWED
0xC0E20011L	TLR_E_CMDTBL_INVALID_DEVICE_ADDRESS
0xC0E20012L	TLR_E_CMDTBL_INVALID_UNIT_ID
0xC0E20013L	TLR_E_CMDTBL_INVALID_FUNCTION_CODE
0xC0E20014L	TLR_E_CMDTBL_INVALID_DATA_COUNT
0xC0E20015L	TLR_E_CMDTBL_INVALID_COMMAND_OFFSET
0xC0E20016L	TLR_E_CMDTBL_INVALID_TRIGGER_TYPE
0xC0E20017L	TLR_E_CMDTBL_INVALID_CYCLE_PERIOD
0xC0E20018L	TLR_E_CMDTBL_INVALID_RESERVED_VALUE
0xC0E20019L	TLR_E_CMDTBL_ONE_TABLE_PER_DEVICE_ADDR
0xC0E2001AL	TLR_E_CMDTBL_OFFSET_NOT_ALIGNED
0xC0E2001BL	TLR_E_CMDTBL_INVALID_DATA_ADDRESS
0xC0E2001CL	TLR_E_CMDTBL_INVALID_BITFIELD_OFFSET
0xC0E2001DL	TLR_E_CMDTBL_INVALID_COMMAND_DELAY
0xC0E2001EL	TLR_E_CMDTBL_INVALID_SCAN_DELAY
0xC06E0070	TLR_E_MODBUS_COMMAND_CONFIG_SIZE Invalid Config Table size, read from the DBM file
0xC06E0071	TLR_E_MODBUS_COMMAND_CONFIG_VERSION Invalid Config Version, read from the DBM file
0xC06E0072	TLR_E_MODBUS_COMMAND_CONFIG_PROT_CLASS Invalid Protocol Class, read from the DBM file
0xC06E0073	TLR_E_MODBUS_COMMAND_CONFIG_DELAY Invalid Delay value, read from the DBM file
0xC06E0074	TLR_E_MODBUS_COMMAND_RESERVED_VALUE The reserved fields must be zero.
0xC06E0075	TLR_E_MODBUS_COMMAND_CMD_SIZE Invalid Command Table size

Hexadecimal value	Definition / description
0xC06E0076	TLR_E_MODBUS_COMMAND_CMD_VERSION Invalid Command Table version
0xC06E0077	TLR_E_MODBUS_COMMAND_CMD_DEVICE Invalid Device ID
0xC06E0078	TLR_E_MODBUS_COMMAND_CMD_UNIT Invalid Unit ID, in MBRTU it must be zero
0xC06E0079	TLR_E_MODBUS_COMMAND_CMD_FUNC_CODE Invalid Function code
0xC06E007A	TLR_E_MODBUS_COMMAND_CMD_SUB_ADDR Invalid values for SubAddress1 & 2, these are reserved
0xC06E007B	TLR_E_MODBUS_COMMAND_CMD_CYCLIC_TIME Invalid Cyclic time
0xC06E007C	TLR_E_MODBUS_COMMAND_CMD_DATA_COUNT Invalid Data Count
0xC06E007D	TLR_E_MODBUS_COMMAND_CMD_DATA_ADDR Invalid Data Address
0xC06E007E	TLR_E_MODBUS_COMMAND_CMD_DPM_ADDR Invalid DPM address
0xC06E007F	TLR_E_MODBUS_COMMAND_CMD_TRIGGER Invalid Trigger
0xC06E0080	TLR_E_MODBUS_COMMAND_NO_CHANGE The selected task instance is not valid
0xC06E0081	TLR_E_MODBUS_COMMAND_INF_MEMORY Insufficient Memory
0xC06E0082	TLR_E_MODBUS_COMMAND_INSTANCE The selected task instance is not valid
0xC06E0083	TLR_E_MODBUS_EMPTY_LIST The database file from the host is empty
0xC06E0084	TLR_E_MODBUS_COMMAND_CMD_DPM_LIMIT_PASSED The command table entries are over the DPM Limit

Table 75: Status/Error codes command table

7 Appendix

7.1 List of tables

Table 1: List of revisions 4
Table 2: Technical data Open Modbus/TCP 6
Table 3: Firmware/stack available for netX 6
Table 4: Configuration 6
Table 5: Additional features 7
Table 6: Terms, abbreviations and definitions 8
Table 7: Function codes 16
Table 8: Example FC 01 17
Table 9: Example FC 02 18
Table 10: Example FC 03 19
Table 11: Example FC 04 20
Table 12: Example FC 05 21
Table 13: Example FC 06 22
Table 14: Example FC 07 23
Table 15: Exception status 23
Table 16: Example FC 15 24
Table 17: Example FC 16 25
Table 18: Example FC 23 26
Table 19: Configuration sequence 31
Table 20: OMB_OMBTASK_CMD_SET_CONFIGURATION_REQ 36
Table 21: Basic parameter: Structure OMB_OMBTASK_CONFIG_T 39
Table 22: TCP/IP Parameter 39
Table 23: Parameter ulFlags 40
Table 24: Parameter ulFlags 41
Table 25: Extended Parameter – Structure 42
Table 26: Open Modbus/TCP Ident configuration data 43
Table 27: OMB_OMBTASK_CMD_SET_CONFIGURATION_CNF –Confirmation of Provide Warmstart Parameters Packet 44
Table 28: Common rcX packets 46
Table 29: Input and Output Data Images 47
Table 30: OMB_OMBTASK_CMD_RECEIVE_IND – Receive Data Indication 54
Table 31: OMB_OMBTASK_CMD_RECEIVE_IND - Packet length 55
Table 32: OMB_OMBTASK_CMD_RECEIVE_RES– Receive Data Response 56
Table 33: OMB_OMBTASK_CMD_RECEIVE_RES - Packet length 57
Table 34: Example: Writing Data via FC6 - Indication 57
Table 35: Example: Writing Data via FC6 – Response 58
Table 36: OMB_OMBTASK_CMD_CONNECTION_IND – Connection Indication Packet 60
Table 37: OMB_OMBTASK_CMD_CONNECTION_RES – Connection Response Packet 61
Table 38: OMB_OMBTASK_CMD_SEND_REQ - Send Data Request 66
Table 39: OMB_OMBTASK_CMD_SEND_REQ - Packet length 67
Table 40: OMB_OMBTASK_CMD_SEND_CNF - Send Data Confirmation 69
Table 41: OMB_OMBTASK_CMD_SEND_CNF - Packet length 70
Table 42: Example: Reading data via FC3 - Request 70
Table 43: Example: Reading data via FC3 - Confirmation 71
Table 44: List of CMD task commands 73
Table 45: CMDTBL_INIT_REQ_T packet 74
Table 46: CMDTBL_INIT_CNF_T packet 75
Table 47: CMDTBL_ADD_TABLE_REQ_T packet 77
Table 48: CMDTBL_ADD_TABLE_CNF_T packet 77
Table 49: CMDTBL_DELET_TABLE_REQ_T packet 78
Table 50: CMDTBL_DELETE_TABLE_CNF_T packet 79
Table 51: CMDTBL_ACTIVATE_TABLE_REQ_T packet 80
Table 52: CMDTBL_ACTIVATE_TABLE_CNF_T packet 81
Table 53: CMDTBL_DEACTIVATE_TABLE_REQ_T packet 82
Table 54: CMDTBL_DEACTIVATE_TABLE_CNF_T packet 83
Table 55: CMDTBL_ADD_COMMAND_REQ_T packet 85
Table 56: CMDTBL_ADD_COMMAND_CNF_T packet 86
Table 57: CMDTBL_DELETE_COMMAND_REQ_T packet 87
Table 58: CMDTBL_DELETE_COMMAND_CNF_T packet 88
Table 59: CMDTBL_ACTIVATE_COMMAND_REQ_T packet 89
Table 60: CMDTBL_ACTIVATE_COMMAND_CNF_T packet 90
Table 61: CMDTBL_DEACTIVATE_COMMAND_REQ_T packet 91
Table 62: CMDTBL_ACTIVATE_COMMAND_CNF_T packet 92

Table 63: CMDTBL_TRIGGER_COMMAND_REQ_T packet 93
Table 64: CMDTBL_TRIGGER_COMMAND_CNF_T packet 94
Table 65: CMDTBL_GET_IO_INFO_REQ_T packet 95
Table 66: CMDTBL_GET_IO_INFO_CNF_T packet 96
Table 67: CMDTBL_DEINIT_REQ packet 97
Table 68: CMDTBL_DEINIT_CNF packet 97
Table 69: CMDTBL_START_STOP_REQ packet 98
Table 70: Start-up Parameters of the OMB-Task 104
Table 71: Start-up Parameters of the OMB_AP-Task 105
Table 72: Start-up Parameters of the CMD-Task 106
Table 73: Status/Error codes OMB task 110
Table 74: Status/Error codes OMB_AP task 111
Table 75: Status/Error codes command table 113

7.2 List of figures

Figure 1: Client/Server principle 11
Figure 2: Structure of Open Modbus/TCP Firmware 12
Figure 3: Host application accesses the Open Modbus/TCP 29
Figure 4: Configuration sequence 30
Figure 5: Command table configuration sequence 33
Figure 6: Addressing model of Open Modbus/TCP in IO mode (default mapping) 48
Figure 7: Addressing model of Open Modbus/TCP in IO mode (alternative mapping) 49
Figure 8: Sequence diagram OMB_OMBTASK_CMD_RECEIVE_IND 50
Figure 9: Sequence diagram OMB_OMBTASK_CMD_CONNECTION_IND 59
Figure 10: Sequence diagram OMB_OMBTASK_CMD_SEND_REQ 62
Figure 11: Example command table 72
Figure 12: Timing diagram for table cyclic time 101
Figure 13: Timing diagram for command cyclic period 102

7.3 Contacts

Headquarters

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

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

Subsidiaries

China
Hilscher 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