Packet API

netX Dual-Port Memory

Packet-based services

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC161001API01EN | Revision 1 | English | 2017-11 | Released | Public

Table of content

1 Introduction 4
1.1 About this document 4
1.2 List of revisions 4
1.3 Terms, abbreviations and definitions 5
1.4 References to documents 5
1.5 Information and data security 5

2 Packet-based services 6
2.1 General packet structure 7
2.2 Recommended packet handling 9
2.3 Additional Packet Data Information 10

3 System services 11
3.1 Function overview 11
3.2 Firmware / System Reset 13
3.3 Identifying netX Hardware 14
3.3.1 Read Hardware Identification Data 15
3.4 Read Hardware Information 18
3.5 Identifying Channel Firmware 20
3.6 System Channel Information Blocks 24
3.6.1 Read System Information Block 25
3.6.2 Read Channel Information Block 26
3.6.3 Read System Control Block 29
3.6.4 Read System Status Block 30
3.7 MAC Address Handling 31
3.7.1 Set MAC Address 32
3.8 Files and Folders 34
3.8.1 List Directories and Files from File System 35
3.8.2 Downloading / Uploading Files 37
3.8.2.1 File Download 38
3.8.2.2 File Download Data 41
3.8.2.3 File Download Abort 43
3.8.3 Uploading Files from netX 44
3.8.3.1 File Upload 45
3.8.3.2 File Upload Data 47
3.8.3.3 File Upload Abort 49
3.8.4 Delete a File 50
3.8.5 Rename a File 52
3.8.6 Creating a CRC32 Checksum 54
3.8.7 Read MD5 File Checksum 55
3.8.8 Read MD5 File Checksum from File Header 57
3.9 Determining the DPM Layout 58
3.10 Security Memory / Flash Device Label 62
3.10.1 Security Memory Zones Content 63
3.10.2 Checksum 64
3.10.3 Zone Read 65
3.10.4 Zone Write 67
3.11 License Information 69
3.12 System Performance Counter 70
3.13 Real-Time Clock 72
3.14 Start RAM based Firmware on netX 75
3.15 Second Stage Bootloader 77
3.15.1 Format the Default Partition 77

4 Communication Channel services 79
4.1 Function overview 79
4.2 Communication Channel Information Blocks 80
4.2.1 Read Common Control Block 80
4.2.2 Read Common Status Block 82
4.2.3 Read Extended Status Block 84
4.3 Read the Communication Flag States 86
4.4 Read I/O Process Data Image Size 88

4.5 Channel Initialization 90
4.6 Delete Protocol Stack Configuration 91
4.7 Lock / Unlock Configuration 92
4.8 Start / Stop Communication 93
4.9 Channel Watchdog Time 94
4.9.1 Get Channel Watchdog Time 94
4.9.2 Set Watchdog Time 95

5 Protocol Stack services 96
5.1 Function overview 96
5.2 Set Handshake Configuration 97
5.3 Modify Configuration Settings 99
5.3.1 Set Parameter Data 102
5.4 Network Connection State 104
5.4.1 Mechanism 104
5.4.2 Obtain List of Slave Handles 106
5.4.3 Obtain Slave Connection Information 108
5.5 Protocol Stack Notifications / Indications 110
5.5.1 Register Application 111
5.5.2 Unregister Application 112
5.6 Link Status Changed Service 113
5.7 Perform a Bus Scan 115
5.8 Get Information about a Fieldbus Device 117
5.9 Configuration in Run 119
5.9.1 Verify Configuration Database 119
5.9.2 Activate Configuration Database 121

6 Status and error codes 122
6.1 Packet error codes 122

7 Appendix 125
7.1 List of figures 125
7.2 List of tables 125
7.3 Legal notes 126
7.4 Contacts 129

1 Introduction

1.1 About this document

The *netX Dual-Port Memory Interface Manual* describes the physical dual-port memory (DPM) layout, content and the general handling procedures and includes the usage of a mailbox system to exchange non-cyclic packet-based data with the firmware and the general definition of packets, packet structures and the handling of command packets and confirmation packets.

This manual

- is an extension to the *netX Dual-Port Memory Interface Manual*,
- defines and describes the non-cyclic packet-based services available in most firmware, and
- focuses on the available system services, their functionality and definitions.

1.2 List of revisions

Rev	Date	Name	Revisions
1	2017-11-27	AM	Splitting DPM Interface Manual into separate manuals for interface and services. Information about a *Flash Device Label* added.

Table 1: List of revisions

1.3 Terms, abbreviations and definitions

Term	Description
DPM	Dual-port memory
FW	Firmware
rcX	Real-time operating system on netX
RTC	Real-time clock

Table 2: Terms, abbreviations and definitions

1.4 References to documents

[1] Hilscher Gesellschaft für Systemautomation mbH: netX Dual-Port Memory Interface Manual, Revision 13, English.

[2] Function Description, Second Stage Boot Loader, netX 10/50/51/52/100/500, V1.4, Revision 14, English.

Table 3: References to documents

1.5 Information and data security

Please take all the usual measures for information and data security, in particular for devices with Ethernet technology. Hilscher explicitly points out that a device with access to a public network (Internet) must be installed behind a firewall or only be accessible via a secure connection such as an encrypted VPN connection. Otherwise the integrity of the device, its data, the application or system section is not safeguarded.

Hilscher can assume no warranty and no liability for damages due to neglected security measures or incorrect installation.

2 Packet-based services

The **Non-Cyclic Data Transfer via Mailboxes using Packets** is the basis for packet-based services. For an explanation and description, see reference [1] that also includes the general packet structure, the packet elements, and the packet exchange with the netX-based firmware.

Structures and definitions

The following C-header files provide structures and definitions used in this document.

rcx_Public.h	Providing the “rcX Packet” structures and function definitions.
rcx_User.h	Providing general error definitions and the netX dual-port memory layout.

If functions are used that are protocol specific, it is necessary to use further header files provided by the protocol stacks.

- TLR header files
  TLR is an abstraction layer using own function, structure and variables definitions partly based on original rcX definitions.
- Protocol-specific header files are coming with the firmware implementation and using additional header files.

2.1 General packet structure

The following structure and descriptions are only a short extract from the information in the *netX Dual-Port Memory Interface Manual*.

Structure Information: *RCX_PACKET*				
Packet Header: *RCX_PACKET_HEADER*				
Area	**Variable**	**Type**	**Value / Range**	**Description**
tHead	ulDest	UINT32	0 ... 0xFFFFFFFF	Destination Address / Handle
	ulSrc	UINT32	0 ... 0xFFFFFFFF	Source Address / Handle
	ulDestId	UINT32	0 ... 0xFFFFFFFF	Destination Identifier
	ulSrcId	UINT32	0 ... 0xFFFFFFFF	Source Identifier
	ulLen	UINT32	0 ... max. packet data size	Packet Data Length (in byte)
	ulId	UINT32	0 ... 0xFFFFFFFF	Packet Identifier
	ulState	UINT32	0 ... 0xFFFFFFFF	Packet State / Error
	ulCmd	UINT32	0 ... 0xFFFFFFFF	Packet Command / Confirmation
	ulExt	UINT32	0 or extension bit mask	Packet Extension
	ulRout	UINT32	0 ... 0xFFFFFFFF	Reserved (routing information)
Packet Data				
Area	**Variable**	**Type**	**Value / Range**	**Description**
abData	...	…	0 … 0xFF	Packet Data (packet specific data)

Table 4: General packet structure

Note: In this document, only the elements which have to be set or changed to create a specific packet are outlined, unchanged elements of the packet are not described.

Destination Address `ulDest / ulDestID` and Source Address `ulSrc / ulSrcID`

These elements are used to address the receiver and sender of a packet.

Packet Data Length `ulLen`

`ulLen` defines how many data counted in bytes follow the packet header. The packet header length is not included in ulLen, because it has a fixed length of 40 bytes (see RCX_PACKET_HEADER)

Packet Identifier `ulId`

`ulId` is intended be used as a unique packet number to destingush between multiple packets of the same type (e.g. multiple packet of the same ulCmd). It is set by the packet creator.

Packet State `ulState`

`ulState` is used to signal packet errors in an answer (response/confirmation) packet. Always zero for command packets (request/indication), because commands with an error are not meaningful.

In answer packets used to signal any problem with the packet header or packet data content (e.g. RCX_E_UNKNOWN_COMMAND, RCX_E_INVALID_PACKET_LEN, RCX_E_PARAMETER_ERROR etc.*)*

Packet Command / Confirmation ulCmd

ulCmd is a predefined code which marks the packet as a command or answer packet. Command codes are defined as even numbers while answers are defined as odd numbers.

Example: Reading the hardware identification of a net-based device

Code	Definition	Description
0x00001EB8	RCX_HW_IDENTIFY_REQ	Command to read general hardware information like device number / serial number etc.
0x00001EB9	RCX_HW_IDENTIFY_CNF	Answer to the RCX_HW_IDENTIFY_REQ command

Packet Extension ulExt

ulExt is used to mark packets as packets of a sequence, in case a transfer consists of multiple packets (e.g. file download).

Reserved (routing information) ulRout

This is reserved for further use (shall not be changed by the receiver of a packet).

Packet Data abData

abData defines the start of the user data area (payload) of the packet. The data content depends on the command or answer given in ulCmd. Each command and answer has a defined user data content while ulLen defines the number of user data bytes contained in the packet.

2.2 Recommended packet handling

- Only one process should handle a mailbox, because multiple processes, accessing the same mailbox, are able to steal packets from each other.
- Receive packet handling should be done before the send packet handling, helping to prevent buffer underruns inside the netX firmware (packet buffers in the firmware are limited).
- A command packet buffer should be initialized with 0 before filled with data.
- `ulId` of each command packet should be unique allowing to follow up the packet execution.
- The receive packet buffer should have the maximum packet size to be able to store a packet with the maximum size. Packet execution on the netX firmware is not serialized and therefore it is unpredictable which packet will be received next if multiple packets are active.
- An answer packet should always be checked against the command packet to be sure to received the requested information. The order of receive packets is not guaranteed when multiple send command are activated. The following elements should be compared.

Send Packet		Receive Packet
`ulCmd`	<->	`ulCmd & RCX_MSK_PACKET_ANSWER`
`ulId`	<->	`ulId`
`ulSrc`	<->	`ulSrc`
`ulSrcId`	<->	`ulSrcId`

- **Note:** The answer code is defined as "command code +1" therefore the lowest bit must be masked out if compared.
- Always check `ulState` of the answer packet to be 0 before evaluating the packet data, `ulState` unequal to 0 signals a packet error.

2.3 Additional Packet Data Information

Packet data are always depend on the command / answer code given in `ulCmd`.

Some of the packet data structures are containing elements where the element length has to be defined / obtained from another element in the structure.

Example: MD5 request with a null terminated file name in the structure

```
Typedef struct RCX_FILE_GET_MD5_REQ_DATA_Ttag
{
  UINT32     ulChannelNo;         /* channel number                          */
  UINT16     usFileNameLength;    /* length of NULL-terminated file name     */

  /* a NULL-terminated file name will follow here */
} RCX_FILE_GET_MD5_REQ_DATA_T;

typedef struct RCX_FILE_GET_MD5_REQ_Ttag
{
  PACKET_HEADER                   tHead;       /* packet header              */
  RCX_FILE_GET_MD5_REQ_DATA_T     tData;       /* packet data                */
} RCX_FILE_GET_MD5_REQ_T;
```

The structure does not cotain an element `szFileName`. The comment inside the structure explains this behaviour, and the length of the filename is given in `usFileNameLength`.

If such an element should be filled out, the filename in this case has to be placed right behind the length parameter `ulFileNameLength`.

```
RCX_PACKET                    tPacket;
RCX_FILE_GET_MD5_REQ_DATA_T* ptMD5Data  = (RCX_FILE_GET_MD5_REQ_DATA_T*)tPacket.abData;
UINT8*                        szFileName = “config.nxd”

memset(&tPacket, 0, sizeof(tPacket));
```

Initialize the packet structure elements:

```
/* set the “normal” fields */
ptMD5Data->ulChannelNo      = 0;
ptMD5Data->usFileNameLength = strlen(szFilename)+1;
```

Append the subsequent information (e.g. file name):

```
/* append the file name*/
strcpy((UINT8*)(ptMD5Data + 1), szFileName);
```

Packet data is also available as lists of elements. Depending to the command, such lists are either defined by a starting data element given the number of elements in the subsequent packet data area or must be calculated by using the packet data length `ulLen`.

3 System services

The operating system (rcX) of the device and the middleware components of the firmware offer **system services**. Most of the functions are common to all netX-based devices. Differences are possible if a device does not offer all common hardware components e.g. Ethernet interface, Security Memory, or file system etc.

3.1 Function overview

System services	Command definition	Page
Reset		
Firmware and system reset	RCX_FIRMWARE_RESET_REQ	13
Identification and information		
Read the general hardware identification information	RCX_HW_IDENTIFY_REQ	14
Read the device-specific hardware information	RCX_HW_HARDWARE_INFO_REQ	18
Read the name and version or firmware, operating system or protocol stack running on a communication channel	RCX_FIRMWARE_IDENTIFY_REQ	20
System Channel Information Blocks		
Read the system channel *System Information Block*	RCX_SYSTEM_INFORMATION_BLOCK_REQ	25
Read the system channel *Channel Information Block*	RCX_CHANNEL_INFORMATION_BLOCK_REQ	26
Read the system channel *System Control Block*	RCX_SYSTEM_CONTROL_BLOCK_REQ	29
Read the system channel *System Status Block*	RCX_SYSTEM_STATUS_BLOCK_REQ	30
MAC Address Handling		
Set / store a new MAC address on the device	RCX_SET_MAC_ADDR_REQ	31
Files and folders		
List directories and files from the file system	RCX_DIR_LIST_REQ	35
Download a file (start, send file data, abort)	RCX_FILE_DOWNLOAD_REQ	38
	RCX_FILE_DOWNLOAD_DATA_REQ	41
	RCX_FILE_DOWNLOAD_ABORT_REQ	43
File Upload (start, read file data, abort)	RCX_FILE_UPLOAD_REQ	45
	RCX_FILE_UPLOAD_DATA_REQ	47
	RCX_FILE_UPLOAD_ABORT_REQ	49
File Delete	RCX_FILE_DELETE_REQ	50
File Rename	RCX_FILE_RENAME_REQ	52
Create a CRC32 checksum	(example code)	54
Calculate the MD5 checksum for a given file	RCX_FILE_GET_MD5_REQ	55
Read the MD5 checksum from the file header of a given file	RCX_FILE_GET_HEADER_MD5_REQ	57

License Information		
Read the license information stored on the netX hardware	RCX_HW_LICENSE_INFO_REQ	69
Determining the DPM Layout		
Read and evaluate the DPM Layout of the system / communication channels	RCX_DPM_GET_BLOCK_INFO_REQ	58
Security Memory / Flash Device Label		
Read device-specific data from Security Memory / Flash Device Label	RCX_SECURITY_EEPROM_READ_REQ	65
Write device-specific data to Security Memory / Flash Device Label	RCX_SECURITY_EEPROM_WRITE_REQ	67
System Performance Counter		
Read the firmware performance counters	RCX_GET_PERF_COUNTERS_REQ	70
Real-Time Clock		
Read / set the real-time clock if available	RCX_TIME_COMMAND_REQ	72
Start RAM based Firmware		
Start a RAM-based firmware which was downloaded before	RCX_CHANNEL_INSTANTIATE_REQ	75
Second Stage Bootloader (only)		
Format the default partition containing the file system	RCX_FORMAT_REQ	77

3.2 Firmware / System Reset

A **Firmware / System Reset** resets the entire netX target.

Firmware Reset request

The application uses the following packet in order to reset the netX chip. The application has to send this packet through the system mailbox.

Structure Information: *RCX_FIRMWARE_RESET_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	8	Packet data length (in Bytes)
ulCmd	UINT32	0x00001E00	RCX_FIRMWARE_RESET_REQ
Data			
ulTimeToReset	UINT32	0	Time delay until reset is executed in milliseconds [ms] Fix: 500ms (not changeable)
ulResetMode	UINT32	0	Reset Mode (not used, set to zero)

Packet structure reference

```
/* CHANNEL RESET REQUEST */
#define RCX_FIRMWARE_RESET_REQ              0x00001E00

typedef struct RCX_FIRMWARE_RESET_REQ_DATA_Ttag
{
  UINT32  ulTimeToReset;  /* time to reset in ms   */
  UINT32  ulResetMode;    /* reset mode parameter  */
} RCX_FIRMWARE_RESET_REQ_DATA_T;

typedef struct RCX_FIRMWARE_RESET_REQtag
{
  RCX_PACKET_HEADER             tHead;  /* packet header */
  RCX_FIRMWARE_RESET_REQ_DATA_T  tData;  /* packet data   */
} RCX_FIRMWARE_RESET_REQ_T;
```

Firmware Reset confirmation

Structure Information: *RCX_FIRMWARE_RESET_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / error code, see Section 6.
ulCmd	UINT32	0x00001E01	RCX_CHANNEL_RESET_CNF

Packet structure reference

```
/* CHANNEL RESET CONFIRMATION */
#define RCX_CHANNEL_RESET_CNF               RCX_CHANNEL_RESET_REQ+1

typedef struct RCX_FIRMWARE_RESET_CNF_Ttag
{
  RCX_PACKET_HEADER    tHead;  /* packet header                          */
} RCX_FIRMWARE_RESET_CNF_T;
```

3.3 Identifying netX Hardware

Hilscher netX-based products uses a **Security Memory** or a **Flash Device Label** to store certain hardware and product-related information that helps to identify the hardware.

The firmware reads the information during a power-up reset and copies certain entries into the *System Information Block* of the system channel located in the dual-port memory.

A configuration tool like SYCON.net evaluates the information and use them to decide whether a firmware file can be downloaded or not. If the information in the firmware file does not match the information read from the dual-port memory, the attempt to download will be rejected.

The following fields are relevant to identify netX hardware.

- Device Number, Device Identification
- Serial Number
- Manufacturer
- Device Class
- Hardware Assembly Options
- Production Date
- License Code

Dual-Port Memory Default Values

In case, the Security Memory or Flash Device Label is not present or provides inconsistent data, the firmware initializes the system information block with the following default data:

- Device Number, Device Identification — Set to zero
- Serial Number — Set to zero
- Manufacturer — Set to UNDEFINED
- Device Class — Set to UNDEFINED
- Hardware Assembly Options — Set to NOT AVAILABLE
- Production Date — Set to zero for both, production year and week
- License Code — Set to zero

3.3.1 Read Hardware Identification Data

The command returns the device number, hardware assembly options, serial number and revision information of the netX hardware. The request packet is passed through the system mailbox only.

Hardware Identify Request

The application uses the following packet in order to read netX hardware information.

Structure Information: *RCX_HW_IDENTIFY_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001EB8	RCX_HW_IDENTIFY_REQ

Packet structure reference

```
/* IDENTIFY FIRMWARE REQUEST */
#define RCX_HW_IDENTIFY_REQ                  0x00001EB8

typedef struct RCX_HW_IDENTIFY_REQ_Ttag
{
  RCX_PACKET_HEADER     tHead;            /* packet header               */
} RCX_HW_IDENTIFY_REQ_T;
```

Hardware Identify Confirmation

The channel firmware returns the following packet.

Structure Information: *RCX_HW_IDENTIFY_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	36 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EB9	RCX_HW_IDENTIFY_CNF
Data			
ulDeviceNumber	UINT32	0 … 0xFFFFFFFF	Device Number
ulSerialNumber	UINT32	0 … 0xFFFFFFFF	Serial Number
ausHwOptions[4]	UINT16	0 … 0xFFFF	Hardware Assembly Option
usDeviceClass	UINT16	0 … 0xFFFF	netX Device Class
bHwRevision	UINT8	0 … 0xFF	Hardware Revision Index
bHwCompatibility	UINT8	0 … 0xFF	Hardware Compatibility Index
ulBootType	UINT32	0 … 8	Hardware Boot Type
ulChipType	UINT32	0 … n	Chip Type (see tables below)
ulChipStep	UINT32	0 … 0x000000FF	Chip Step
ulRomcodeRevision	UINT32	0 … 0x00000FFF	ROM Code Revision

Packet structure reference

```
/* HARDWARE IDENTIFY CONFIRMATION */
#define RCX_HW_IDENTIFY_CNF                 RCX_HW_IDENTIFY_REQ+1

typedef struct RCX_HW_IDENTIFY_CNF_DATA_Ttag
{
  UINT32   ulDeviceNumber;              /* device number / identification */
  UINT32   ulSerialNumber;              /* serial number                  */
  UINT16   ausHwOptions[4];             /* hardware options               */
  UINT16   usDeviceClass;               /* device class                   */
  UINT8    bHwRevision;                 /* hardware revision              */
  UINT8    bHwCompatibility;            /* hardware compatibility         */
  UINT32   ulBootType;                  /* boot type                      */
  UINT32   ulChipTyp;                   /* chip type                      */
  UINT32   ulChipStep;                  /* chip step                      */
  UINT32   ulRomcodeRevision;           /* rom code revision              */
} RCX_HW_IDENTIFY_CNF_DATA_T;

typedef struct RCX_HW_IDENTIFY_CNF_Ttag
{
  RCX_PACKET_HEADER           tHead;    /* packet header                  */
  RCX_HW_IDENTIFY_CNF_DATA_T  tData;    /* packet data                    */
} RCX_HW_IDENTIFY_CNF_T;
```

Note: The tables *Boot Type* and *Chip Type* are only an excerpt of the avialable options. The complete option list can be found in the "rcX_User.h" file

Boot Type

This field indicates how the netX operating system was started.

Value	Definition / Description
0x00000000	ROM Loader: PARALLEL FLASH (SRAM Bus)
0x00000001	ROM Loader: PARALLEL FLASH (Extension Bus)
0x00000002	ROM Loader: DUAL-PORT MEMORY
0x00000003	ROM Loader: PCI INTERFACE
0x00000004	ROM Loader: MULTIMEDIA CARD
0x00000005	ROM Loader: I²C BUS
0x00000006	ROM Loader: SERIAL FLASH
0x00000007	2nd Stage Boot Loader: SERIAL FLASH
0x00000008	2nd Stage Boot Loader: RAM
Other values are reserved	

Table 5: Boot Type

Chip Type

This field indicates the type of chip that is used.

Value	Definition / Description
0x00000000	Unknown
0x00000001	netX 500
0x00000002	netX 100
0x00000003	netX 50
0x00000004	netX 10
0x00000005	netX 51
Other values are reserved	

Table 6: Chip Type

3.4 Read Hardware Information

Hardware Info Request

Obtain information about the netX hardware. The packet is send through the system mailbox.

Structure Information: *RCX_HW_HARDWARE_INFO_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001EF6	RCX_HW_HARDWARE_INFO_REQ

Packet structure reference

```
/* READ HARDWARE INFORMATION REQUEST */
#define RCX_HW_HARDWARE_INFO_REQ             0x00001EF6

typedef struct RCX_HW_HARDWARE_INFO_REQ_Ttag
{
  RCX_PACKET_HEADER     tHead;             /* packet header               */
} RCX_HW_HARDWARE_INFO_REQ_T;
```

Hardware Info Confirmation

Structure Information: *RCX_HW_HARDWARE_INFO_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	56 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EF7	RCX_HW_HARDWARE_INFO_CNF
Data			
ulDeviceNumber	UINT32	0 … 0xFFFFFFFF	Device Number / Identification
ulSerialNumber	UINT32	0 … 0xFFFFFFFF	Serial Number
ausHwOptions[4]	Array of UINT16	0 … 0xFFFF	Hardware Assembly Option
usManufacturer	UINT16	0 … 0xFFFF	Manufacturer Code
usProductionDate	UINT16	0 … 0xFFFF	Production Date
ulLicenseFlags1	UINT32	0 … 0xFFFFFFFF	License Flags 1
ulLicenseFlags2	UINT32	0 … 0xFFFFFFFF	License Flags 2
usNetxLicenseID	UINT16	0 … 0xFFFF	netX License Identification
usNetxLicenseFlags	UINT16	0 … 0xFFFF	netX License Flags
usDeviceClass	UINT16	0 … 0xFFFF	netX Device Class
bHwRevision	UINT8	0 …0xFFFF	Hardware Revision Index
bHwCompatibility	UINT8	0	Hardware Compatibility Index
ulHardware Features1	UINT32	0	Hardware Features 1 (not used, set to 0)
ulHardware Features2	UINT32	0	Hardware Features 2 (not used, set to 0)
bBootOption	UINT8	0	Boot Option (not used, set to 0)
bReserved[11]	UINT8	0	Reserved, set to 0

Packet structure reference

```
/* READ HARDWARE INFORMATION CONFIRMATION */
#define RCX_HW_HARDWARE_INFO_CNF            RCX_HW_HARDWARE_INFO_REQ+1

typedef struct RCX_HW_HARDWARE_INFO_CNF_DATA_Ttag
{
  UINT32  ulDeviceNumber;         /* device number              */
  UINT32  ulSerialNumber;         /* serial number              */
  UINT16  ausHwOptions[4];        /* hardware assembly options  */
  UINT16  usManufacturer;         /* device manufacturer        */
  UINT16  usProductionDate;       /* production date            */
  UINT32  ulLicenseFlags1;        /* license flags 1            */
  UINT32  ulLicenseFlags2;        /* license flags 2            */
  UINT16  usNetxLicenseID;        /* license ID                 */
  UINT16  usNetxLicenseFlags;     /* license flags              */
  UINT16  usDeviceClass;          /* device class               */
  UINT8   bHwRevision;            /* hardware revision          */
  UINT8   bHwCompatibility;       /* hardware compatibility     */
  UINT32  ulHardwareFeatures1;    /* not used, set to 0         */
  UINT32  ulHardwareFeatures2;    /* not used, set to 0         */
  UINT8   bBootOption;            /* not used, set to 0         */
  UINT8   bReserved[11];          /* reserved, set to 0         */
} RCX_HW_HARDWARE_INFO_CNF_DATA_T;

typedef struct RCX_HW_HARDWARE_INFO_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;  /* packet header     */
  RCX_HW_HARDWARE_INFO_CNF_DATA_T  tData;  /* packet data       */
} RCX_HW_HARDWARE_INFO_CNF_T;
```

3.5 Identifying Channel Firmware

This request returns the name, version and date of the boot loader, operating system, firmware or protocol stack running on the netX chip. The information depends on the kind of executed firmware (boot loader or protocol firmware) and on which mailbox the request is passed to the system.

Depending on the mailbox (system / communication channel) also the destination address *ulDest* and the *ulChannelID* parameter within the packet are used to define the returned information.

Deliviered versions information accordingto the mailbox, *ulDest* and *ulChannelID*:

Firmware: *System Channel Mailbox*		
ulDest	**ulChannelID**	**Returned Information**
RCX_PACKET_DEST_SYSTEM	0xFFFFFFFF	Version of the RCX operating system
	0 ... 3	Protokol stack name of the communication channel given by *ulChannelID*
RCX_PACKET_DEST_DEFAULT_CHANNEL	0xFFFFFFFF	Don't care Firmware name (see note below)
	0 ... 3	Protokol stack name of the communication channel given by *ulChannelID*
Firmware: *Communication Channel Mailbox*		
ulDest	**ulChannelID**	**Returned Information**
RCX_PACKET_DEST_SYSTEM	0xFFFFFFFF	Version of the RCX operating system
	0 ... 3	Protokol stack name of the communication channel given by *ulChannelID*
RCX_PACKET_DEST_DEFAULT_CHANNEL	0xFFFFFFFF	Firmware name (see note below)
	0..3	Don't care Protokol stack name of the selected communication channel
Bootloader: *System Channel Mailbox*		
ulDest	**ulChannelID**	**Returned Information**
RCX_PACKET_DEST_SYSTEM or RCX_PACKET_DEST_DEFAULT_CHANNEL	ignored	Bootloader version

Note: Usually ***Firmware Name*** and ***ProtocolStack Name*** of communication channel 0 are equal

Note: Version information delivered back depends on the channel where the command is initiated, the receiver of the packet *ulDest* and the value given in *ulChannelId*.

Firmware Identify Request

Depending on the requirements, the packet is passed through the system mailbox to obtain operating system information, or it is passed through the channel mailbox to obtain protocol stack related information.

Structure Information: *RCX_FIRMWARE_IDENTIFY_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000 0x00000020	RCX_PACKET_DEST_SYSTEM RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001EB6	RCX_FIRMWARE_IDENTIFY_REQ
Data			
ulChannelId	UINT32	see definition above	Channel Identification

Packet structure reference

```
/* IDENTIFY FIRMWARE REQUEST */
#define RCX_FIRMWARE_IDENTIFY_REQ           0x00001EB6

/*Channel Identification */
#define RCX_SYSTEM_CHANNEL                  0xFFFFFFFF
#define RCX_COMM_CHANNEL_0                  0x00000000
#define RCX_COMM_CHANNEL_1                  0x00000001
#define RCX_COMM_CHANNEL_2                  0x00000002
#define RCX_COMM_CHANNEL_3                  0x00000003
```

```
typedef struct RCX_FIRMWARE_IDENTIFY_REQ_DATA_Ttag
{
  UINT32     ulChannelId;                   /* channel ID                */
} RCX_FIRMWARE_IDENTIFY_REQ_DATA_T;

typedef struct RCX_FIRMWARE_IDENTIFY_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
  RCX_FIRMWARE_IDENTIFY_REQ_DATA_T tData;   /* packet data               */
} RCX_FIRMWARE_IDENTIFY_REQ_T;
```

Firmware Identify Confirmation

The channel firmware returns the following packet.

Structure Information: *RCX_FIRMWARE_IDENTIFY_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	 76 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EB7	RCX_FIRMWARE_IDENTIFY_CNF
Data			
tFwVersion	Structure	see below	Firmware Version
tFwName	Structure	see below	Firmware Name
tFwDate	Structure	see below	Firmware Date

Packet structure reference

```
/* IDENTIFY FIRMWARE CONFIRMATION */
#define RCX_FIRMWARE_IDENTIFY_CNF           RCX_FIRMWARE_IDENTIFY_REQ+1

typedef struct RCX_FW_IDENTIFICATION_Ttag
{
  RCX_FW_VERSION_T  tFwVersion;           /* firmware version            */
  RCX_FW_NAME_T     tFwName;              /* firmware name               */
  RCX_FW_DATE_T     tFwDate;              /* firmware date               */
} RCX_FW_IDENTIFICATION_T;

typedef struct RCX_FIRMWARE_IDENTIFY_CNF_DATA_Ttag
{
  RCX_FW_IDENTIFICATION_T   tFirmwareIdentification;    /* firmware ID    */
} RCX_FIRMWARE_IDENTIFY_CNF_DATA_T;

typedef struct RCX_FIRMWARE_IDENTIFY_CNF_Ttag
{
  RCX_PACKET_HEADER                 tHead;  /* packet header             */
  RCX_FIRMWARE_IDENTIFY_CNF_DATA_T  tData;  /* packet data               */
} RCX_FIRMWARE_IDENTIFY_CNF_T;
```

Version *tFwVersion*

The version information field consist of four parts separated into a *Major*, *Minor*, *Build* and *Revision* section.

- *Major* number, given in hexadecimal format [0..0xFFFF].
  The number is increased for significant enhancements in functionality (backward compatibility cannot be assumed)
- *Minor* number, given in hexadecimal format [0..0xFFFF].
  The the number is incremented when new features or enhancements have been added (backward compatibility is intended).
- *Build* number, given in hexadecimal format [0..0xFFFF].
  The number denotes bug fixes or a new firmware build
- R*evision* number, given in hexadecimal format [0..0xFFFF].
  It is used to signal hotfixes for existing versions. It is set to zero for new *Major* / *Minor* / *Build* updates.

Version Structure

```
typedef struct RCX_FW_VERSION_Ttag
{
  UINT16              usMajor;            /* major version number */
  UINT16              usMinor;            /* minor version number */
  UINT16              usBuild;            /* build number         */
  UINT16              usRevision;         /* revision number      */
} RCX_FW_VERSION_T;
```

Name *tFwName*

This field holds the name of the firmware comprised of ASCII characters.

- *bNameLength* holds the length of valid bytes in the *abName[63]* array.
- *abName[63]* contains the firmware name as ASCII characters, limited to 63 characters

Firmware Name Structure:

```
typedef struct RCX_FW_NAME_Ttag
{
  UINT8               bNameLength;        /* length of firmware name    */
  UINT8               abName[63];         /* firmware name              */
} RCX_FW_NAME_T;
```

Date *tFwDate*

The ***tFwDate*** field holds the date of the firmware release.

- *usYear* year is given in hexadecimal format in the range [0..0xFFFF]
- *bMonth* month is given in hexadecimal format in the range [0x01..0x0C]
- *bDay* day is given in hexadecimal format in the range [0x01..0x1F].

Firmware Date Structure:

```
typedef struct RCX_FW_DATE_Ttag
{
  UINT16              usYear;             /* firmware creation year    */
  UINT8               bMonth;             /* firmware creation month   */
  UINT8               bDay;               /* firmware creation day     */
} RCX_FW_DATE_T;
```

3.6 System Channel Information Blocks

The following packets are defined to make system data blocks available for read access through the mailbox channel. These packets are used by configuration tools, like SYCON.net, if they are connected via a serial interface and need to read these informtion from the netX hardware.

If the requested data block exceeds the maximum mailbox size, the block is transferred in a sequenced or fragmented manner (see *netX Dual-Port Memory Interface Manual* for details about fragmented packet transfer).

Available Blocks:

Block Name	DPM Structure	Description
System Information Block	NETX_SYSTEM_INFO_BLOCK	Contains general infromation of the hardware (device) like the cookie, device number, serial number etc.
Channel Information Block	NETX_CHANNEL_INFO_BLOCK	Contains infromations about the available channels in a firmware
System Control Block	NETX_SYSTEM_CONTROL_BLOCK	Contains available control registers and flags to control the hardware
System Status Block	NETX_SYSTEM_STATUS_BLOCK	Contains state information about the hardware (e.g. Boot Error, System Error, CPU Load information etc.)

3.6.1 Read System Information Block

The packet outlined in this section is used to request the *System Information Block*. Therefore it is passed through the system mailbox.

System Information Block Request

Structure Information: *RCX_READ_SYS_INFO_BLOCK_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001E32	RCX_SYSTEM_INFORMATION_BLOCK_REQ

Packet structure reference

```
/* READ SYSTEM INFORMATION BLOCK REQUEST */
#define RCX_SYSTEM_INFORMATION_BLOCK_REQ     0x00001E32

typedef struct RCX_READ_SYS_INFO_BLOCK_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header            */
} RCX_READ_SYS_INFO_BLOCK_REQ_T;
```

System Information Block Confirmation

The following packet is returned.

Structure Information: *RCX_READ_SYS_INFO_BLOCK_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	48 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E33	RCX_SYSTEM_INFORMATION_BLOCK_CNF
Data			
tSystemInfo	Structure		System Information Block See *netX Dual-Port Memory Interface Manual* for more details.

Packet structure reference

```
/* READ SYSTEM INFORMATION BLOCK CONFIRMATION */
#define RCX_SYSTEM_INFORMATION_BLOCK_CNF     RCX_SYSTEM_INFORMATION_BLOCK_REQ+1

typedef struct RCX_READ_SYS_INFO_BLOCK_CNF_DATA_Ttag
{
  NETX_SYSTEM_INFO_BLOCK               tSystemInfo; /* packet data       */
} RCX_READ_SYS_INFO_BLOCK_CNF_DATA_T;

typedef struct RCX_READ_SYS_INFO_BLOCK_CNF_Ttag
{
  RCX_PACKET_HEADER                  tHead;   /* packet header          */
  RCX_READ_SYS_INFO_BLOCK_CNF_DATA_T  tData;   /* packet data            */
} RCX_READ_SYS_INFO_BLOCK_CNF_T;
```

3.6.2 Read Channel Information Block

The packet outlined in this section is used to request the *Channel Information Block*. Therefore it is passed through the system mailbox. There is one packet for each of the channels. The channels are identified by their channel ID or port number. The total number of blocks is part of the structure of the Channel Information Block of the system channel.

Channel Information Block Request

This packet is used to request the *Channel Information Block* (*NETX_CHANNEL_INFO_BLOCK*) of a channel specified by *ulChannelId*.

Structure Information: *RCX_READ_CHNL_INFO_BLOCK_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001E34	RCX_CHANNEL_INFORMATION_BLOCK_REQ
Data			
ulChannelId	UINT32	0 … 7	Channel Identifier Port Number, Channel Number

Packet structure reference

```
/* READ CHANNEL INFORMATION BLOCK REQUEST */
#define RCX_CHANNEL_INFORMATION_BLOCK_REQ    0x00001E34

typedef struct RCX_READ_CHNL_INFO_BLOCK_REQ_DATA_Ttag
  UINT32 ulChannelId;                          /* channel id               */
} RCX_READ_CHNL_INFO_BLOCK_REQ_DATA_T;

typedef struct RCX_READ_CHNL_INFO_BLOCK_REQ_Ttag
{
  RCX_PACKET_HEADER                   tHead;  /* packet header            */
  RCX_READ_CHNL_INFO_BLOCK_REQ_DATA_T tData;  /* packet data              */
} RCX_READ_CHNL_INFO_BLOCK_REQ_T;
```

Channel Information Block Confirmation

The confirmation packet contains the *tChannelInfo* data structure which is defined as a union of multiple structures. To be able to us the data, the first element of any union structure defines the channel type. This type must be evaluated before the corresponding structure can be used to evealuate the content of the structure.

Structure Information: *RCX_READ_CHNL_INFO_BLOCK_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT 32	16 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT 32	See Below	Status / Error Code, see Section 6
ulCmd	UINT 32	0x00001E3 5	RCX_CHANNEL_INFORMATION_BLOCK_CNF
Data			
tChannel Info	Struct ure		Channel Information Block See *netX Dual-Port Memory Interface Manual* for more details.

Packet structure reference

```
/* READ CHANNEL INFORMATION BLOCK CONFIRMATION */
#define RCX_CHANNEL_INFORMATION_BLOCK_CNF    RCX_CHANNEL_INFORMATION_BLOCK_REQ+1

typedef union NETX_CHANNEL_INFO_BLOCKtag
{
  NETX_SYSTEM_CHANNEL_INFO            tSystem;
  NETX_HANDSHAKE_CHANNEL_INFO         tHandshake;
  NETX_COMMUNICATION_CHANNEL_INFO     tCom;
  NETX_APPLICATION_CHANNEL_INFO       tApp;
} NETX_CHANNEL_INFO_BLOCK;

typedef struct RCX_READ_CHNL_INFO_BLOCK_CNF_DATA_Ttag
{
  NETX_CHANNEL_INFO_BLOCK           tChannelInfo;  /* channel info block   */
} RCX_READ_CHNL_INFO_BLOCK_CNF_DATA_T;

typedef struct RCX_READ_CHNL_INFO_BLOCK_CNF_Ttag
{
  RCX_PACKET_HEADER                    tHead;     /* packet header        */
  RCX_READ_CHNL_INFO_BLOCK_CNF_DATA_T  tData;     /* packet data          */
} RCX_READ_CHNL_INFO_BLOCK_CNF_T;
```

Example how to evaluate the structure

```
uint32_t                ulBlockID
NETX_CHANNEL_INFO_BLOCK* ptChannel;

/* Iterate over all block definitions, start with channel 0 information */
ptChannel = &tChannelInfo

/*-----------------------------------------------------------*/
/* Evaluate the channel information blockt                   */
/*-----------------------------------------------------------*/
for(ulBlockID = 0 ulBlockID < NETX_MAX_SUPPORTED_CHANNELS; ++ulBlockID)
{
  /* Check Block types */
  switch(ptChannel->tSystem.bChannelType))
  {
    case RCX_CHANNEL_TYPE_COMMUNICATION:
    {
      /* This is a communication channel, read an information */
      uint16_t usActualProtocolClass;
      usActualProtocolClass = ChannelInfo->tCom.usProtocolClass;
    }
    break;

    case RCX_CHANNEL_TYPE_APPLICATION:
      /* This is an application channel */
    break;

    case RCX_CHANNEL_TYPE_HANDSHAKE:
      /* This is the handshake channel containing the handshake registers */
      break;

    case RCX_CHANNEL_TYPE_SYSTEM:
      /* This is the system channel */
      break;

    case RCX_CHANNEL_TYPE_UNDEFINED:
    case RCX_CHANNEL_TYPE_RESERVED:
   default:
      /* Do not process these types */
    break;

  } /* end switch */
  ++ptChannel; /* address next infromation from the channel info block */

} /* end for loop */
```

3.6.3 Read System Control Block

System Control Block Request

This packet is used to request the *System Control Block (NETX_SYSTEM_CONTROL_BLOCK*).

Structure Information: *RCX_READ_SYS_CNTRL_BLOCK_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001E36	RCX_SYSTEM_CONTROL_BLOCK_REQ

Packet structure reference

```
/* READ SYSTEM CONTROL BLOCK REQUEST */
#define RCX_SYSTEM_CONTROL_BLOCK_REQ         0x00001E36

typedef struct RCX_READ_SYS_CNTRL_BLOCK_REQ_Ttag
{
  RCX_PACKET_HEADER                 tHead;   /* packet header             */
} RCX_READ_SYS_CNTRL_BLOCK_REQ_T;
```

System Control Block Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_READ_SYS_CNTRL_BLOCK_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	8 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E37	RCX_SYSTEM_CONTROL_BLOCK_CNF
Data			
tSystem Control	Structure		System Control Block See *netX Dual-Port Memory Interface Manual* for more details.

Packet structure reference

```
/* READ SYSTEM CONTROL BLOCK CONFIRMATION */
#define RCX_SYSTEM_CONTROL_BLOCK_CNF         RCX_SYSTEM_CONTROL_BLOCK_REQ+1

typedef struct RCX_READ_SYS_CNTRL_BLOCK_CNF_DATA_Ttag
{
  NETX_SYSTEM_CONTROL_BLOCK         tSystemControl;
} RCX_READ_SYS_CNTRL_BLOCK_CNF_DATA_T;

typedef struct RCX_READ_SYS_CNTRL_BLOCK_CNF_Ttag
{
  RCX_PACKET_HEADER                     tHead;     /* packet header        */
  RCX_READ_SYS_CNTRL_BLOCK_CNF_DATA_T   tData;     /* packet data          */
} RCX_READ_SYS_CNTRL_BLOCK_CNF_T;
```

3.6.4 Read System Status Block

System Status Block Request

This packet is used to request the *System Status Block* (*NETX_SYSTEM_STATUS_BLOCK*)

Structure Information: *RCX_READ_SYS_STATUS_BLOCK_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001E38	RCX_SYSTEM_STATUS_BLOCK_REQ

Packet structure reference

```
/* READ SYSTEM STATUS BLOCK REQUEST */
#define RCX_SYSTEM_STATUS_BLOCK_REQ          0x00001E38

typedef struct RCX_READ_SYS_STATUS_BLOCK_REQ_Ttag
{
  RCX_PACKET_HEADER                 tHead;   /* packet header             */
} RCX_READ_SYS_STATUS_BLOCK_REQ_T;
```

System Status Block Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_READ_SYS_STATUS_BLOCK_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	 64 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E39	RCX_SYSTEM_STATUS_BLOCK_CNF
Data			
tSystemState	Structure		System Status Block See *netX Dual-Port Memory Interface Manual* for more details.

Packet structure reference

```
/* READ SYSTEM STATUS BLOCK CONFIRMATION */
#define RCX_SYSTEM_STATUS_BLOCK_CNF          RCX_SYSTEM_STATUS_BLOCK_REQ+1

typedef struct RCX_READ_SYS_STATUS_BLOCK_CNF_DATA_Ttag
{
  NETX_SYSTEM_STATUS_BLOCK              tSystemState;
} RCX_READ_SYS_STATUS_BLOCK_CNF_DATA_T;

typedef struct RCX_READ_SYS_STATUS_BLOCK_CNF_Ttag
{
  RCX_PACKET_HEADER                     tHead;     /* packet header        */
  RCX_READ_SYS_STATUS_BLOCK_CNF_DATA_T  tData;     /* packet data          */
} RCX_READ_SYS_STATUS_BLOCK_CNF_T;
```

3.7 MAC Address Handling

Any *Ethernet* based hardware requires a MAC address which makes the device unique identifyable. A netX based device may offer up to 4 *Ethernet* interfaces where each of the interfaces requires an own unique MAC address.

Usually the MAC address is stored on the device and it is not changable. In a netX environment the MAC address will be stored in a *Security Memory* or *Flash Device Label* if available.

Unfortunately the space in the *Security Memory* / *Flash Device Label* is very limited and a netX device can **only store ONE MAC** address permanetly.

Any necessary, additional, MAC address will be generated by incremeting the "default" stored MAC address.

In other words, depending on the used protocol stacks and system layout a netX target system may need more than one MAC addresses assigned.
The first address is stored on the system, additional addresses are created from the first one.

ATTENTION **A netX *Ethernet* based firmware will use up to 4 MAC addresses.**

The first MAC address is usually stored on the hardware.
3 additional, subsequent, MAC addresses will be used by the firmware, created by incrementing the first one.

Ethernet Port 0: stored MAC address
Ethernet Port 1: stored MAC address **+ 1**
Ethernet Port 2: stored MAC address **+ 2**
Ethernet Port 3: stored MAC address **+ 3**

This means up to 4 MAC addresses are occupied by a netX device.

Device without a Security Memory / Flash Device Label

If the hardware does not offer a *Security Memory* or *FLASH Device Label* to store the MAC address (this could happen on slave devices), a fieldbus protocol stack waits for the host application to provide a MAC address before proceeding with the fieldbus system initialization.

On such a system, the MAC address must be set on each system start or power cycle.

3.7.1 Set MAC Address

This service can be used to either set a MAC address permant if a *Security Memory* or *Flash Device Label* is available or temporarily if not.

- **Hardware without Security Memory / Flash Device Label**
  The MAC address is stored temporarily and lost after a reset or power cycle. Neither the *Store* flag nor the *Force* flag has a meaning.
- **Hardware with Security Memory / Flash Device Label**
  - **No MAC address stored or MAC address set to 0**
    If the *Store* flag is set, the MAC address is written and stored permanently.
  - **MAC address already stored**
    Both, the *Store* flag and the *Force* flag have to be set in order to overwrite the stored MAC address and to store the new address permanently.

Set MAC Address Request

The following packet can be used to set a MAC address for the netX system. The packet is send through the *System Channel* and handled by the netX firmware.

Structure Information: *RCX_SET_MAC_ADDR_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	12	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001EEE	RCX_SET_MAC_ADDR_REQ
Data			
ulParam	UINT32	0x00000001 0x00000002	Parameter Field (see below) RCX_STORE_MAC_ADDRESS RCX_FORCE_MAC_ADDRESS
abMacAddr[6]	UINT8		MAC Address
abPad[2]	UINT8	0x00	Padding bytes, set to zero

Packet structure reference

```
/* SET MAC ADDRESS REQUEST */
#define RCX_SET_MAC_ADDR_REQ                0x00001EEE

#define RCX_STORE_MAC_ADDRESS               0x00000001
#define RCX_FORCE_MAC_ADDRESS               0x00000002

typedef struct RCX_SET_MAC_ADDR_REQ_DATA_Ttag
{
  UINT32  ulParam;                      /* parameter bit field          */
  UINT8   abMacAddr[6];                 /* MAC address                  */
  UINT8   abPad[2];                     /* pad bytes, set to zero       */
} RCX_SET_MAC_ADDR_REQ_DATA_T;

typedef struct RCX_SET_MAC_ADDR_REQ_Ttag
{
  RCX_PACKET_HEADER               tHead;       /* packet header          */
  RCX_SET_MAC_ADDR_REQ_DATA_T     tData;       /* packet data            */
} RCX_SET_MAC_ADDR_REQ_T;
```

Parameter Field: *ulParam*

31	30	…	12	11	10	9	8	7	6	5	4	3	2	1	0	Bit Number
															Store MAC Address *RCX_STORE_MAC_ADDRESS*	
														Force MAC Address *RCX_FORCE_MAC_ADDRESS*		
Reserved, set to zero																

Table 7: Set MAC Address Parameter Field

Bit No.	Definition	Definition / Description
0	*RCX_STORE_MAC_ADDRESS*	*Store* MAC Address This flag needs to be set if a MAC address shall be written and stored. Storing the value is only possible if the previous value of the MAC address is empty or set to 0. Otherwise an error code is returned. On success, the MAC address is stored permanently. The flag is ignored if no *Security Memory* / *Flash Device Label* is available.
1	*RCX_FORCE_MAC_ADDRESS*	*Force* MAC Address This flag needs to be set together with the *Store MAC Address* flag in order to overwrite and store an MAC address. On success, the MAC address is stored permanently. The flag is ignored if no *Security Memory* / *Flash Device Label* available.
2 … 31	none	Reserved, set to 0

Table 8: Set MAC Address Parameter Field

Set MAC Address Confirmation

The system channel returns the following packet.

Structure Information: *RCX_SET_MAC_ADDR_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Codes, see Section 6
ulCmd	UINT32	0x00001EEF	RCX_SET_MAC_ADDR_CNF

Packet structure reference

```
/* SET MAC ADDRESS CONFIRMATION */
#define RCX_SET_MAC_ADDR_CNF                RCX_SET_MAC_ADDR_REQ+1

typedef struct RCX_SET_MAC_ADDR_CNF_Ttag
{
  RCX_PACKET_HEADER              tHead;       /* packet header          */
} RCX_SET_MAC_ADDR_CNF_T;
```

3.8 Files and Folders

A standard netX firmware contains a file system or storage meachnism which holds firmware, configuration and user files. To be able to access these files, the following services are offered.

Note	The file system which is used in the netX firmware is FAT based and supports only file names in the "8.3" format.

Note	File download / upload can be handled via the system mailbox or via a channel mailbox. In both cases, the destination identifier has to be *ulDest* = RCX_SYSTEM_CHANNEL. The difference between the system mailbox and a communication channel mailbox is just the size of the packet length which can be transferred.

The netX firmware acknowledges each of the packets and may return an error code in the confirmation, if a failure occurs.

3.8.1 List Directories and Files from File System

Directories and files in the rcX file system can be listed by the command outlined below. The default file system layout is shown below.

File System Layout

Volume	Directory	Description
root	System	unused / internal use
	PORT_0	Communication Channel 0
	PORT_1	Communication Channel 1
	PORT_2	Communication Channel 2
	PORT_3	Communication Channel 3

Note: A netX firmware is always stored in the sub-directory of *Port 0.*

Directory List Request

Structure Information: *RCX_DIR_LIST_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	6 + n	sizeof(RCX_DIR_LIST_REQ_DATA_T) + strlen("DirName")+1 Remark: 0 can be used for the second, third, etc. packet.
ulCmd	UINT32	0x00001E70	RCX_DIR_LIST_REQ
ulExt	UINT32	0x00, 0xC0	0x00: for the first packet. 0xC0: for the next packets.
Data			
ulChannelNo	UINT32	0 … 3 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 System Channel
usDirName Length	UINT16	n	Name Length Length of the Directory Name (in Bytes) strlen("DirName")+1
	UINT8	ASCII	Directory Name ASCII string, zero terminated e.g. "\\PORT_0", "\", etc.

Packet structure reference

```
/* DIRECTORY LIST REQUEST */
#define RCX_DIR_LIST_REQ                    0x00001E70

/* Channel Number */
#define RCX_COMM_CHANNEL_0                  0x00000000
#define RCX_COMM_CHANNEL_1                  0x00000001
#define RCX_COMM_CHANNEL_2                  0x00000002
#define RCX_COMM_CHANNEL_3                  0x00000003
```

```
typedef struct RCX_DIR_LIST_REQ_DATA_Ttag
{
  UINT32    ulChannelNo;        /* 0 = channel 0 ... 3 = channel 3             */
                                /* 0xFFFFFFFF = system, see RCX_FILE_xxxx      */
  UINT16    usDirNameLength;    /* length of NULL terminated string            */
  /* a NULL-terminated name string will follow here */
} RCX_DIR_LIST_REQ_DATA_T;

typedef struct RCX_DIR_LIST_REQ_Ttag
{
  RCX_PACKET_HEADER           tHead;          /* packet header               */
  RCX_DIR_LIST_REQ_DATA_T     tData;          /* packet data                 */
} RCX_DIR_LIST_REQ_T;
```

Directory List Confirmation

Structure Information: *RCX_DIR_LIST_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	 24 0 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK If ulState = RCX_S_OK and ulExt = 0x40 (last packet) Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E71	RCX_DIR_LIST_CNF
ulExt	UINT32	0x80, 0xC0, 0x40	0x80 for the first packet. 0xC0 for the following packets. 0x40 for the last packet.
Data			
szName[16]	UINT8		File Name
ulFileSize	UINT32	m	File Size in Bytes
bFileType	UINT8	 0x00000001 0x00000002	File Type RCX_DIR_LIST_CNF_FILE_TYPE_DIRECTORY RCX_DIR_LIST_CNF_FILE_TYPE_FILE
bReserved	UINT8	0	Reserved, unused
usReserved2	UINT16	0	Reserved, unused

Packet structure reference

```
/* DIRECTORY LIST CONFIRMATION */
#define RCX_DIR_LIST_CNF                    RCX_DIR_LIST_REQ+1

/* TYPE: DIRECTORY */
#define RCX_DIR_LIST_CNF_FILE_TYPE_DIRECTORY 0x00000001

/* TYPE: FILE */
#define RCX_DIR_LIST_CNF_FILE_TYPE_FILE     0x00000002

typedef struct RCX_DIR_LIST_CNF_DATA_Ttag
{
  UINT8                   szName[16];   /* file name                    */
  UINT32                  ulFileSize;   /* file size                    */
  UINT8                   bFileType;    /* file type                    */
  UINT8                   bReserved;    /* reserved, set to 0           */
  UINT16                  bReserved2    /* reserved, set to 0           */
} RCX_DIR_LIST_CNF_DATA_T;

typedef struct RCX_DIR_LIST_CNF_Ttag
{
  RCX_PACKET_HEADER       tHead;        /* packet header                */
  RCX_DIR_LIST_CNF_DATA_T tData;        /* packet data                  */
} RCX_DIR_LIST_CNF_T;
```

3.8.2 Downloading / Uploading Files

Any download / upload of files to/from the netX firmware is handled via netX packets as described below. The netX operating system (rcX) creates a file system where the files are stored.

To download a file, the user application has to split the file into smaller pieces that fit into a packet data area and send them to the netX. Similar handling is necessary for a file upload, where a file can only be requested in pieces which have to be assembled by the user application.

For file uploads / downloads (e.g. firmware or configuration files) where the data does not fit into a single packet, the packet header field *ulExt* in conjunction with the packet identifier *ulID* has to be used to control packet sequence handling, indicating the first, last and sequenced packets.

Note: The user application must send/request files in the order of its original sequence. The *ulId* field in the packet holds a sequence number. It starts with 0 and is incremented for each new request packet.
Sequence numbers shall not be skipped or used twice because the netX firmware **cannot** re-assemble file data received out of order.

Example:

Single Packet Upload/Download			**Two Packet Upload/Download**		
Definition	**ulId**	**ulExt**	**Definition**	**ulId**	**ulExt**
RCX_FILE_DOWNLOAD_REQ	0	RCX_PACKET_SEQ_NONE	RCX_FILE_DOWNLOAD_REQ	0	RCX_PACKET_SEQ_NONE
RCX_FILE_DOWNLOAD_DATA_REQ	1	RCX_PACKET_SEQ_NONE	RCX_FILE_DOWNLOAD_DATA_REQ	1	RCX_PACKET_SEQ_FIRST
			RCX_FILE_DOWNLOAD_DATA_REQ	2	RCX_PACKET_SEQ_LAST

Multi Packet Upload/Download		
Definition	**ulId**	**ulExt**
RCX_FILE_DOWNLOAD_REQ	0	RCX_PACKET_SEQ_NONE
RCX_FILE_DOWNLOAD_DATA_REQ	1	RCX_PACKET_SEQ_FIRST
.....	.....	.....
RCX_FILE_DOWNLOAD_DATA_REQ	ulId +1	RCX_PACKET_SEQ_MIDDLE
RCX_FILE_DOWNLOAD_DATA_REQ	ulId +1	RCX_PACKET_SEQ_MIDDLE
.....	.....	.....
RCX_FILE_DOWNLOAD_DATA_REQ	ulId +1	RCX_PACKET_SEQ_LAST

3.8.2.1 File Download

The download procedure starts with a *File Download Request* packet. The user application provides at least the file length and file name.

The system responds with the maximum packet data size, which can be used in the subsequent *File Download Data* packets. The application has to transfer the entire file by sending as many data packets as necessary.

Each packet will be confirmed by the firmware. The download is finished with the last packet.

Flowchart

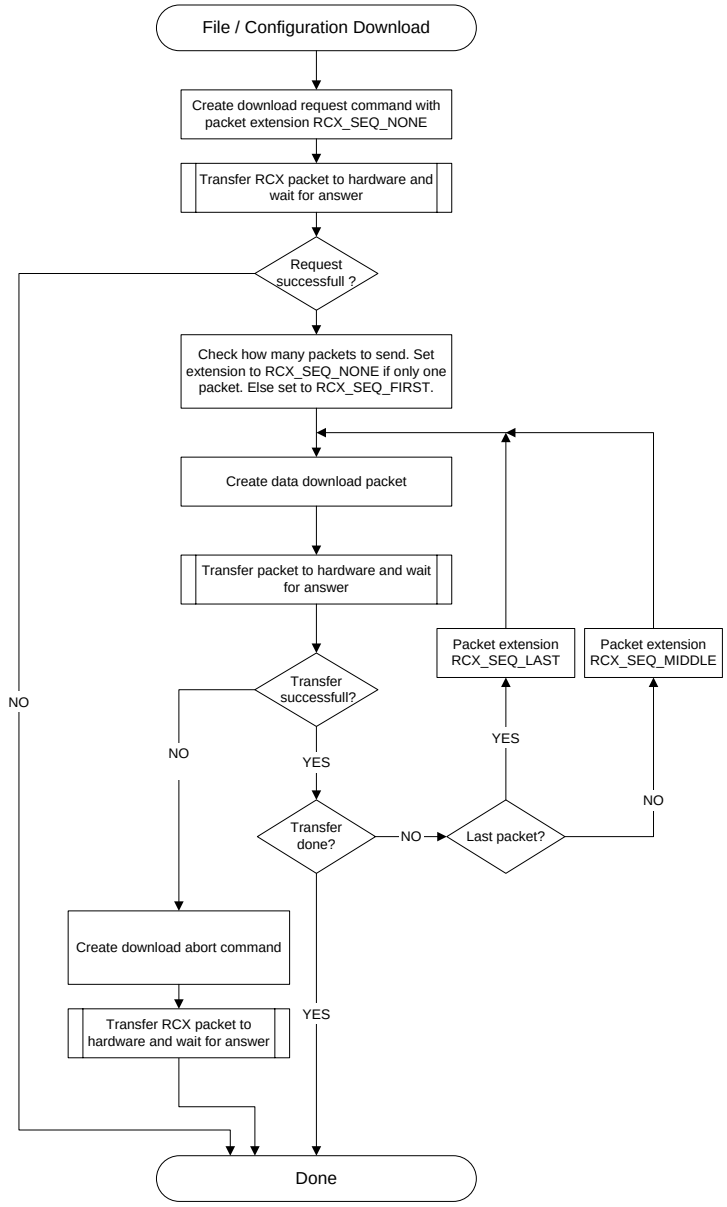

Figure 1: Flowchart File Download

Note: If an error occurs during the download, the process must be canceled by sending a *File Download Abort* command.

File Download Request

The packet below is the first request to be sent to the netX firmware to start a file download. The application provides the length of the file and its name in the request packet.

Structure Information: *RCX_FILE_DOWNLOAD_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_SYSTEM_CHANNEL
ulLen	UINT32	18 + n	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001E62	RCX_FILE_DOWNLOAD_REQ
ulId	UINT32	0	Packet Identifer
ulExt	UINT32	0x00000000	Extension RCX_PACKET_SEQ_NONE
Data			
ulXferType	UINT32	0x00000001	Download Transfer Type RCX_FILE_XFER_FILE
ulMaxBlockSize	UINT32	1 … m	Max Block Size Maximum Size of Block per Packet
ulFileLength	UINT32	n	File size to be downloaded in bytes
ulChannelNo	UINT32	0 … 3 0xFFFFFFFF	Destination Channel Number Communication Channel 0 … 3 System Channel
usFileNameLength	UINT16	n	Length of Name Length of the following file name (in Bytes)
(file name)	UINT8	ASCII	File Name ASCII string, zero terminated

Packet structure reference

```
/* FILE DOWNLOAD REQUEST */
#define RCX_FILE_DOWNLOAD_REQ                0x00001E62

/* TRANSFER FILE */
#define RCX_FILE_XFER_FILE                   0x00000001

/* TRANSFER INTO FILE SYSTEM */
#define RCX_FILE_XFER_FILESYSTEM             0x00000001

/* TRANSFER MODULE */
#define RCX_FILE_XFER_MODULE                 0x00000002

/* Channel Number */
#define RCX_SYSTEM_CHANNEL                   0xFFFFFFFF
#define RCX_COMM_CHANNEL_0                   0x00000000
#define RCX_COMM_CHANNEL_1                   0x00000001
#define RCX_COMM_CHANNEL_2                   0x00000002
#define RCX_COMM_CHANNEL_3                   0x00000003

typedef struct RCX_FILE_DOWNLOAD_REQ_DATA_Ttag
{
  UINT32   ulXferType;
  UINT32   ulMaxBlockSize;
  UINT32   ulFileLength;
  UINT32   ulChannelNo;
  UINT16   usFileNameLength;
  /* a NULL-terminated file name follows here                          */
  /* UINT8   abFileName[ ];                                            */
} RCX_FILE_DOWNLOAD_REQ_DATA_T;
```

```
typedef struct RCX_FILE_DOWNLOAD_REQ_Ttag
{
  RCX_PACKET_HEADER              tHead;   /* packet header               */
  RCX_FILE_DOWNLOAD_REQ_DATA_T   tData;   /* packet data                 */
} RCX_FILE_DOWNLOAD_REQ_T;
```

File Download Confirmation

The netX firmware acknowledges the request with the following confirmation packet. It contains the size of the data block that can be transferred in one packet.

Structure Information: *RCX_FILE_DOWNLOAD_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	4 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E63	RCX_FILE_DOWNLOAD_CNF
Data			
ulMaxBlockSize	UINT32	1 … n	Max Block Size Maximum Size of Block per Packet

Packet structure reference

```
/* FILE DOWNLOAD CONFIRMATION */
#define RCX_FILE_DOWNLOAD_CNF                RCX_FILE_DOWNLOAD_REQ+1

typedef struct RCX_FILE_DOWNLOAD_CNF_DATA_Ttag
{
  UINT32  ulMaxBlockSize;
} RCX_FILE_DOWNLOAD_CNF_DATA_T;

typedef struct RCX_FILE_DOWNLOAD_CNF_Ttag
{
  RCX_PACKET_HEADER              tHead;    /* packet header               */
  RCX_FILE_DOWNLOAD_CNF_DATA_T   tData;    /* packet data                 */
} RCX_FILE_DOWNLOAD_CNF_T;
```

3.8.2.2 File Download Data

This packet is used to transfer a block of data to the netX operating system rcX to be stored in the file system. The term *data block* is used to describe a portion of a file. The data block in the packet is identified by a block or sequence number and is secured through a continuous CRC32 checksum.

Note: If the download fails, the rcX returns an error code in *ulState*. The user application then has to send an *Abort File Download Request* packet (see page 43) and start over.

The block or sequence number *ulBlockNo* starts with zero for the first data packet and is incremented by one for each following packet. The checksum in *ulChksum* is calculated as a CRC32 polynomial. It is calculated continuously over all data packets that were sent already. A sample on how to calculate the checksum is included in this manual.

File Download Data Request

Structure Information: *RCX_FILE_DOWNLOAD_DATA_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_SYSTEM_CHANNEL
ulLen	UINT32	8 + n	Packet Data Length (in Bytes)
ulId	UINT32	n	Packet Identifier
ulCmd	UINT32	0x00001E64	RCX_FILE_DOWNLOAD_DATA_REQ
ulId	UINT32	ulId+1	Packet Identifer **Note:** Should be incremented for each request
ulExt	UINT32	0x00000000 0x00000080 0x000000C0 0x00000040	Extension RCX_PACKET_SEQ_NONE (if data fits into one packet) RCX_PACKET_SEQ_FIRST RCX_PACKET_SEQ_MIDDLE RCX_PACKET_SEQ_LAST
Data			
ulBlockNo	UINT32	0 ... m	Block Number Block or Sequence Number
ulChksum	UINT32	S	Checksum CRC32 Polynomial
	UINT8	0 … 0xFF	File Data Block (length given in *ulLen*)

Packet structure reference

```
/* FILE DOWNLOAD DATA REQUEST*/
#define RCX_FILE_DOWNLOAD_DATA_REQ          0x00001E64

/* PACKET SEQUENCE */
#define RCX_PACKET_SEQ_NONE                 0x00000000
#define RCX_PACKET_SEQ_FIRST                0x00000080
#define RCX_PACKET_SEQ_MIDDLE               0x000000C0
#define RCX_PACKET_SEQ_LAST                 0x00000040

typedef struct RCX_FILE_DOWNLOAD_DATA_REQ_DATA_Ttag
{
  UINT32  ulBlockNo;               /* block number                      */
  UINT32  ulChksum;                /* cumulative CRC-32 checksum        */
  /* data block follows here                                           */
  /* UINT8   abData[ ];                                                */
} RCX_FILE_DOWNLOAD_DATA_REQ_DATA_T;
```

```
typedef struct RCX_FILE_DOWNLOAD_DATA_REQ_Ttag
{
  RCX_PACKET_HEADER                  tHead;     /* packet header           */
  RCX_FILE_DOWNLOAD_DATA_REQ_DATA_T  tData;     /* packet data             */
} RCX_FILE_DOWNLOAD_DATA_REQ_T;
```

File Download Data Confirmation

The rcX operating system returns the following confirmation packet. It contains the expected CRC32 checksum of the data block.

Structure Information: *RCX_FILE_DOWNLOAD_DATA_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	 4 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E65	RCX_FILE_DOWNLOAD_DATA_CNF
Data			
ulExpected Crc32	UINT32	 S	Checksum Expected CRC32 Polynomial

Packet structure reference

```
/* FILE DOWNLOAD DATA CONFIRMATION */
#define RCX_FILE_DOWNLOAD_DATA_CNF          RCX_FILE_DOWNLOAD_DATA_REQ+1

/* PACKET SEQUENCE */
#define RCX_PACKET_SEQ_NONE                 0x00000000

typedef struct RCX_FILE_DOWNLOAD_DATA_CNF_DATA_Ttag
{
  UINT32  ulExpectedCrc32;          /* expected CRC-32 checksum          */
} RCX_FILE_DOWNLOAD_DATA_CNF_DATA_T;

typedef struct RCX_FILE_DOWNLOAD_DATA_CNF_Ttag
{
  RCX_PACKET_HEADER                  tHead;     /* packet header           */
  RCX_FILE_DOWNLOAD_DATA_CNF_DATA_T  tData;     /* packet data             */
} RCX_FILE_DOWNLOAD_DATA_CNF_T;
```

3.8.2.3 File Download Abort

If an error occurs during the download of a file (*ulState* not equal to RCX_S_OK), the user application has to abort the download procedure by sending the *File Download Abort* command.

This command can also be used by an application to abort the download procedure at any time.

File Download Abort Request

Structure Information: *RCX_FILE_DOWNLOAD_ABORT_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_SYSTEM_CHANNEL
ulCmd	UINT32	0x00001E66	RCX_FILE_DOWNLOAD_ABORT_REQ
ulId	UINT32	ulId+1	Packet Identifer **Note:** Should be incremented for each request

Packet structure reference

```
/* ABORT DOWNLOAD REQUEST */
#define RCX_FILE_DOWNLOAD_ABORT_REQ          0x00001E66

typedef struct RCX_FILE_DOWNLOAD_ABORT_REQ_Ttag
{
  RCX_PACKET_HEADER     tHead;            /* packet header               */
} RCX_FILE_DOWNLOAD_ABORT_REQ_T;
```

File Download Abort Confirmation

The rcX operating system returns the following confirmation packet, indicating that the download was aborted.

Structure Information: *RCX_FILE_DOWNLOAD_ABORT_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E67	RCX_FILE_DOWNLOAD_ABORT_CNF

Packet structure reference

```
/* ABORT DOWNLOAD REQUEST */
#define RCX_FILE_DOWNLOAD_ABORT_CNF          RCX_FILE_DOWNLOAD_ABORT_REQ+1

typedef struct RCX_FILE_DOWNLOAD_ABORT_CNF_Ttag
{
  RCX_PACKET_HEADER     tHead;            /* packet header               */
} RCX_FILE_DOWNLOAD_ABORT_CNF_T;
```

3.8.3 Uploading Files from netX

Just as the download process, the upload process is handled via packets. The file to be uploaded is selected by the file name. During the *File Upload* request, the file name is transferred to the rcX. If the requested file exists, the rcX returns all necessary file information in the response.

The host application creates *File Upload Data* request packets, which will be acknowledged by the rcX with the corresponding confirmation packets holding portions of the file data. The application has to continue sending *File Upload Data* request packets until the entire file is transferred. Receiving the last confirmation packet finishes the upload process.

Flowchart

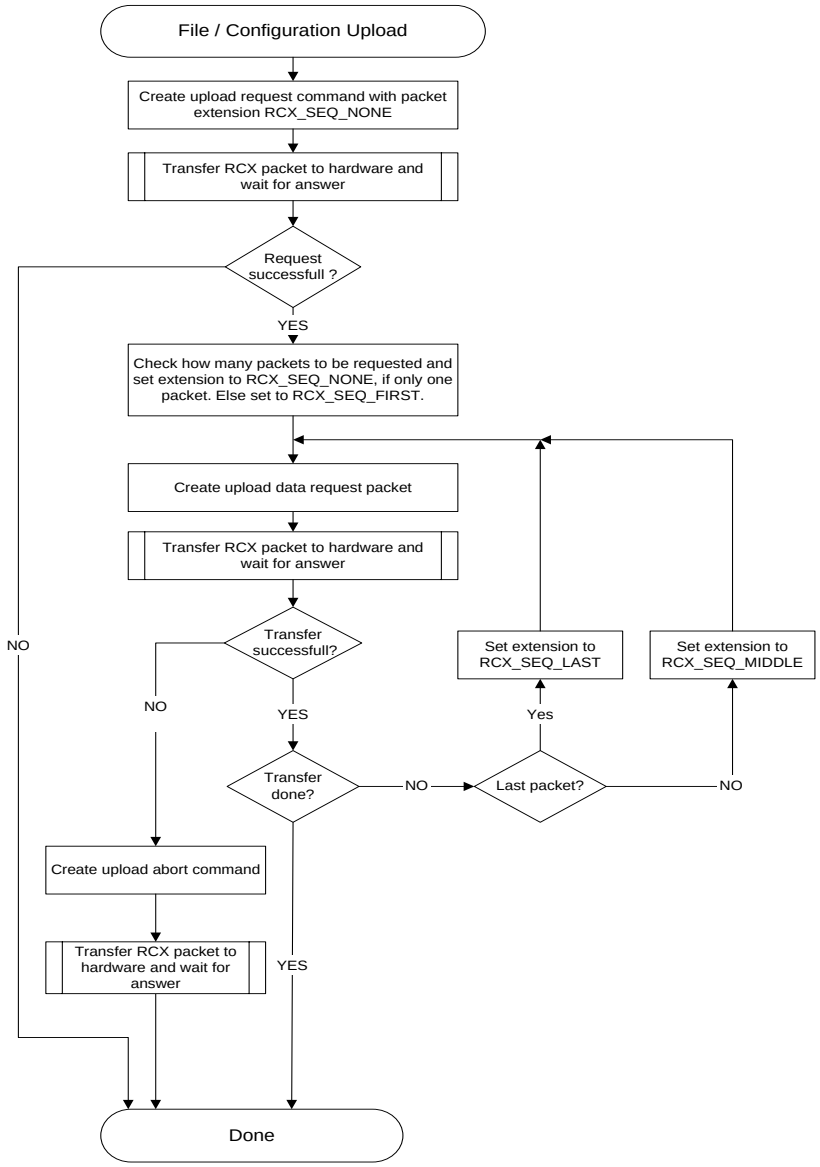

Figure 2: Flowchart File Upload

3.8.3.1 File Upload

Note: If an error occurs during a file upload, the process **must** be canceled by sending a *File Upload Abort* command.

File Upload Request

The file upload request is the first request to be sent to the system. The application provides the length of the file and its name in the request packet.

Structure Information: *RCX_FILE_UPLOAD_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	14 + n	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001E60	RCX_FILE_UPLOAD_REQ
ulId	UINT32	0	Packet Identifier
ulExt	UINT32	0x00000000	Extension RCX_PACKET_SEQ_NONE
Data			
ulXferType	UINT32	0x00000001	Transfer Type: RCX_FILE_XFER_FILE
ulMaxBlockSize	UINT32	1 ... m	Max Block Size Maximum Size of Block per Packet
ulChannelNo	UINT32	0 ... 3 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 System Channel
usFileNameLength	UINT16	n	Length of Name Length of Following File Name (in Bytes)
	UINT8	ASCII	File Name ASCII string, zero terminated

Packet structure reference

```
/* FILE UPLOAD COMMAND */
#define RCX_FILE_UPLOAD_REQ                 0x00001E60

/* PACKET SEQUENCE */
#define RCX_PACKET_SEQ_NONE                 0x00000000

/* TRANSFER TYPE */
#define RCX_FILE_XFER_FILE                  0x00000001

/* CHANNEL Number */
#define RCX_SYSTEM_CHANNEL                  0xFFFFFFFF
#define RCX_COMM_CHANNEL_0                  0x00000000
#define RCX_COMM_CHANNEL_1                  0x00000001
#define RCX_COMM_CHANNEL_2                  0x00000002
#define RCX_COMM_CHANNEL_3                  0x00000003
```

```
typedef struct RCX_FILE_UPLOAD_REQ_DATA_Ttag
{
  UINT32   ulXferType;                    /* transfer type              */
  UINT32   ulMaxBlockSize;                /* block size                 */
  UINT32   ulChannelNo;                   /* channel number             */
  UINT16   usFileNameLength;              /* length of file name        */
  /* a NULL-terminated file name follows here                           */
  /* UINT8   abFileName[ ];                  file name                  */
} RCX_FILE_UPLOAD_REQ_DATA_T;
```

```
typedef struct RCX_FILE_UPLOAD_REQ_Ttag
{
  RCX_PACKET_HEADER           tHead;      /* packet header               */
  RCX_FILE_UPLOAD_REQ_DATA_T  tData;      /* packet data                 */
} RCX_FILE_UPLOAD_REQ_T;
```

File Upload Confirmation

The netX system acknowledges the request with the following confirmation packet.

Structure Information: *RCX_FILE_UPLOAD_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	8 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E61	RCX_FILE_UPLOAD_CNF_T
Data			
ulMaxBlockSize	UINT32	n	Max Block Size Maximum Size of Block per Packet
ulFileLength	UINT32	n	File Length Total File Length (in Bytes)

Packet structure reference

```
/* FILE UPLOAD CONFIRMATION */
#define RCX_FILE_UPLOAD_CNF                 RCX_FILE_UPLOAD_REQ+1

/* PACKET SEQUENCE */
#define RCX_PACKET_SEQ_NONE                 0x00000000

typedef struct RCX_FILE_UPLOAD_CNF_DATA_Ttag
{
  UINT32  ulMaxBlockSize;            /* maximum block size possible     */
  UINT32  ulFileLength;              /* file size to transfer           */
} RCX_FILE_UPLOAD_CNF_DATA_T;

typedef struct RCX_FILE_UPLOAD_CNF_Ttag
{
  RCX_PACKET_HEADER           tHead;  /* packet header                  */
  RCX_FILE_UPLOAD_CNF_DATA_T  tData;  /* packet data                    */
} RCX_FILE_UPLOAD_CNF_T;
```

3.8.3.2 File Upload Data

This packet is used to transfer a block of data from the netX system to the user application. The term *data block* is used to describe a portion of a file. The data block in the packet is identified by a block or sequence number and is secured through a continuous CRC32 checksum.

File Upload Data Request

Structure Information: *RCX_FILE_UPLOAD_DATA_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001E6E	RCX_FILE_UPLOAD_DATA_REQ
ulId	UINT32	ulId+1	Packet Identifer **Note:** Should be incremented for each request
ulExt	UINT32	 0x00000000 0x00000080 0x000000C0 0x00000040	Extension RCX_PACKET_SEQ_NONE (if data fits into one packet) RCX_PACKET_SEQ_FIRST RCX_PACKET_SEQ_MIDDLE RCX_PACKET_SEQ_LAST

Packet structure reference

```
/* FILE UPLOAD DATA REQUEST */
#define RCX_FILE_UPLOAD_DATA_REQ            0x00001E6E

/* PACKET SEQUENCE */
#define RCX_PACKET_SEQ_NONE                 0x00000000
#define RCX_PACKET_SEQ_FIRST                0x00000080
#define RCX_PACKET_SEQ_MIDDLE               0x000000C0
#define RCX_PACKET_SEQ_LAST                 0x00000040

typedef struct RCX_FILE_UPLOAD_DATA_REQ_Ttag
{
  PACKET_HEADER     tHead;                /* packet header               */
} RCX_FILE_UPLOAD_DATA_REQ_T;
```

File Upload Data Confirmation

The confirmation contains the block number and the expected CRC32 checksum of the data block.

Structure Information: *RCX_FILE_UPLOAD_DATA_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
`ulLen`	UINT32	8 + n 0	Packet Data Length (in Bytes) If `ulState` = `RCX_S_OK` Otherwise
`ulState`	UINT32	See Below	Status / Error Code, see Section 6
`ulCmd`	UINT32	0x00001E6F	RCX_FILE_UPLAOD_DATA_CNF
Data			
`ulBlockNo`	UINT32	0 … m	Block Number Block or Sequence Number
`ulChksum`	UINT32	S	Checksum CRC32 Polynomial
	UINT8		File Data Block (Size is n given in *`ulLen`*)

Packet structure reference

```
/* FILE DATA UPLOAD CONFIRMATION */
#define RCX_FILE_UPLAOD_DATA_CNF            RCX_FILE_UPLOAD_DATA_REQ +1

/* PACKET SEQUENCE */
#define RCX_PACKET_SEQ_NONE                 0x00000000

typedef struct RCX_FILE_UPLOAD_DATA_CNF_DATA_Ttag
{
  UINT32  ulBlockNo;                     /* block number starting from 0 */
  UINT32  ulChksum;                      /* cumulative CRC-32 checksum   */
  /* data block follows here                                             */
  /* UINT8   abData[ ];                                                  */
} RCX_FILE_UPLOAD_DATA_CNF_DATA_T;

typedef struct RCX_FILE_UPLOAD_DATA_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
  RCX_FILE_UPLOAD_DATA_CNF_DATA_T  tData;   /* packet data               */
} RCX_FILE_UPLOAD_DATA_CNF_T;
```

Block Number *`ulBlockNo`*

The block number *ulBlockNo* starts with zero for the first data packet and is incremented by one for every following packet. The rcX sends the file in the order of its original sequence. Sequence numbers are not skipped or used twice.

Checksum *`ulChksum`*

The checksum *ulChksum* is calculated as a CRC32 polynomial. It is calculated continuously over all data packets that were sent already. A sample to calculate the checksum is included in the toolkit for netX based products.

3.8.3.3 File Upload Abort

In case of an error (*ulState* not equal to RCX_S_OK) during an upload, the application has to cancel the upload procedure by sending the abort command.

If necessary, the application can use the command abort an upload procedure at any time.

File Upload Abort Request

Structure Information: *RCX_FILE_UPLOAD_ABORT_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001E5E	RCX_FILE_UPLOAD_ABORT_REQ
ulId	UINT32	ulId+1	Packet Identifer **Note:** Should be incremented for each request

Packet structure reference

```
/* FILE ABORT UPLOAD REQUEST */
#define RCX_FILE_UPLOAD_ABORT_REQ           0x00001E5E

typedef struct RCX_FILE_UPLOAD_ABORT_REQ_Ttag
{
  RCX_PACKET_HEADER     tHead;            /* packet header               */
} RCX_FILE_UPLOAD_ABORT_REQ_T;
```

File Upload Abort Confirmation

The system acknowledges an abort command with the following confirmation packet.

Structure Information: *RCX_FILE_UPLOAD_ABORT_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E5F	RCX_FILE_UPLOAD_ABORT_CNF

Packet structure reference

```
/* FILE ABORT UPLOAD CONFIRMATION */
#define RCX_FILE_UPLOAD_ABORT_CNF           RCX_FILE_UPLOAD_ABORT_REQ+1

typedef struct RCX_FILE_UPLOAD_ABORT_CNF_Ttag
{
  RCX_PACKET_HEADER     tHead;            /* packet header               */
} RCX_FILE_UPLOAD_ABORT_CNF_T;
```

3.8.4 Delete a File

If the target hardware supports a FLASH/RAM based file system, all downloaded files like firmware (FLASH only), configuration and user files are stored in the file system.

The following service can be used to delete files from the target files system.

File Delete Request

Structure Information: *RCX_FILE_DELETE_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	6 + n	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001E6A	RCX_FILE_DELETE_REQ
Data			
ulChannelNo	UINT32	0 … 3 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 System Channel
usFileName Length	UINT16	n	Length of Name Length of the Following File Name (in Bytes)
	UINT8	ASCII	File Name ASCII string, zero terminated

Packet structure reference

```
/* FILE DELETE REQUEST */
#define RCX_FILE_DELETE_REQ                  0x00001E6A

/* Channel Number */
#define RCX_SYSTEM_CHANNEL                   0xFFFFFFFF
#define RCX_COMM_CHANNEL_0                   0x00000000
#define RCX_COMM_CHANNEL_1                   0x00000001
#define RCX_COMM_CHANNEL_2                   0x00000002
#define RCX_COMM_CHANNEL_3                   0x00000003

typedef struct RCX_FILE_DELETE_REQ_DATA_Ttag
{
  UINT32      ulChannelNo;          /* 0 = channel 0 ... 3 = channel 3        */
                                    /* 0xFFFFFFFF = system, see RCX_FILE_xxxx */
  UINT16      usFileNameLength;     /* length of NULL-terminated file name    */
  /* a NULL-terminated file name will follow here */
} RCX_FILE_DELETE_REQ_DATA_T;

typedef struct RCX_FILE_DELETE_REQ_Ttag
{
  RCX_PACKET_HEADER            tHead;          /* packet header               */
  RCX_FILE_DELETE_REQ_DATA_T   tData;          /* packet data                 */
} RCX_FILE_DELETE_REQ_T;
```

File Delete Confirmation

Structure Information: *RCX_FILE_DELETE_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E6B	RCX_FILE_DELETE_CNF

Packet structure reference

```
/* FILE DELETE REQUEST */
#define RCX_FILE_DELETE_CNF                 RCX_FILE_DELETE_REQ+1

typedef struct RCX_FILE_DELETE_CNF_Ttag
{
  RCX_PACKET_HEADER       tHead;                /* packet header       */
} RCX_FILE_DELETE_CNF_T;
```

3.8.5 Rename a File

This service can be used to rename files in the target file system.

File Rename Request

Structure Information: *RCX_FILE_RENAME_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	8+m+n	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001E7C	RCX_FILE_RENAME_REQ
Data			
ulChannelNo	UINT32	0 … 3 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 System Channel
usOldNameLength	UINT16	m	Length of Old File Name Length of following NULL terminated old File Name (in Bytes)
usNewNameLength	UINT16	n	Length of New File Name Length of following NULL terminated new File Name (in Bytes)
	UINT8	ASCII	Old File Name ASCII string, zero terminated
	UINT8	ASCII	New File Name ASCII string, zero terminated

Packet structure reference

```
/* FILE RENAME REQUEST */
#define RCX_FILE_RENAME_REQ                0x00001E7C

typedef struct RCX_FILE_RENAME_REQ_DATA_Ttag
{
  UINT32  ulChannelNo;     /* 0..3 = Channel 0..3, 0xFFFFFFFF = System */
  UINT16  usOldNameLength; /* length of NUL-terminated old file name that will follow */
  UINT16  usNewNameLength; /* length of NUL-terminated new file name that will follow */

  /* old NUL-terminated file name will follow here */
  /* new NUL-terminated file name will follow here */
} RCX_FILE_RENAME_REQ_DATA_T;

typedef struct RCX_FILE_RENAME_REQ_Ttag
{
  RCX_PACKET_HEADER          tHead;        /* packet header */
  RCX_FILE_RENAME_REQ_DATA_T tData;        /* packet data */
} RCX_FILE_RENAME_REQ_T;
```

File Rename Confirmation

Structure Information: *RCX_FILE_RENAME_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E7D	RCX_FILE_RENAME_CNF

Packet structure reference

```
/* FILE RENAME CONFIRMATION */
#define RCX_FILE_RENAME_CNF                RCX_FILE_RENAME_REQ+1

typedef struct RCX_FILE_RENAME_CNF_Ttag
{
  RCX_PACKET_HEADER tHead;                 /* packet header */
} RCX_FILE_RENAME_CNF_T;
```

3.8.6 Creating a CRC32 Checksum

This is an example which shows the generation of a CRC32 checksum, necessary for certain file functions like a file download (such an example can also be found in the internet).

```
/*****************************************************************************/
/*! Create a CRC32 value from the given buffer data
*   \param ulCRC continued CRC32 value
*   \param pabBuffer buffer to create the CRC from
*   \param ulLength buffer length
*   \return CRC32 value                                                      */
/*****************************************************************************/
static unsigned long CreateCRC32( unsigned long  ulCRC,
                                  unsigned char* pabBuffer,
                                  unsigned long  ulLength )
{
  if( (0 == pabBuffer) || (0 == ulLength) )
  {
    return ulCRC;
  }
  ulCRC = ulCRC ^ 0xffffffff;
  for(;ulLength > 0; --ulLength)
  {
    ulCRC = (Crc32Table[((ulCRC) ^ (*(pabBuffer++)) ) & 0xff] ^ ((ulCRC) >> 8));
  }
  return( ulCRC ^ 0xffffffff );
}
```

```
/*****************************************************************************/
/*! CRC 32 lookup table                                                      */
/*****************************************************************************/
static unsigned long Crc32Table[256]=
{
  0x00000000UL, 0x77073096UL, 0xee0e612cUL, 0x990951baUL, 0x076dc419UL, 0x706af48fUL, 0xe963a535UL,
  0x9e6495a3UL, 0x0edb8832UL, 0x79dcb8a4UL, 0xe0d5e91eUL, 0x97d2d988UL, 0x09b64c2bUL, 0x7eb17cbdUL,
  0xe7b82d07UL, 0x90bf1d91UL, 0x1db71064UL, 0x6ab020f2UL, 0xf3b97148UL, 0x84be41deUL, 0x1adad47dUL,
  0x6ddde4ebUL, 0xf4d4b551UL, 0x83d385c7UL, 0x136c9856UL, 0x646ba8c0UL, 0xfd62f97aUL, 0x8a65c9ecUL,
  0x14015c4fUL, 0x63066cd9UL, 0xfa0f3d63UL, 0x8d080df5UL, 0x3b6e20c8UL, 0x4c69105eUL, 0xd56041e4UL,
  0xa2677172UL, 0x3c03e4d1UL, 0x4b04d447UL, 0xd20d85fdUL, 0xa50ab56bUL, 0x35b5a8faUL, 0x42b2986cUL,
  0xdbbbc9d6UL, 0xacbcf940UL, 0x32d86ce3UL, 0x45df5c75UL, 0xdcd60dcfUL, 0xabd13d59UL, 0x26d930acUL,
  0x51de003aUL, 0xc8d75180UL, 0xbfd06116UL, 0x21b4f4b5UL, 0x56b3c423UL, 0xcfba9599UL, 0xb8bda50fUL,
  0x2802b89eUL, 0x5f058808UL, 0xc60cd9b2UL, 0xb10be924UL, 0x2f6f7c87UL, 0x58684c11UL, 0xc1611dabUL,
  0xb6662d3dUL, 0x76dc4190UL, 0x01db7106UL, 0x98d220bcUL, 0xefd5102aUL, 0x71b18589UL, 0x06b6b51fUL,
  0x9fbfe4a5UL, 0xe8b8d433UL, 0x7807c9a2UL, 0x0f00f934UL, 0x9609a88eUL, 0xe10e9818UL, 0x7f6a0dbbUL,
  0x086d3d2dUL, 0x91646c97UL, 0xe6635c01UL, 0x6b6b51f4UL, 0x1c6c6162UL, 0x856530d8UL, 0xf262004eUL,
  0x6c0695edUL, 0x1b01a57bUL, 0x8208f4c1UL, 0xf50fc457UL, 0x65b0d9c6UL, 0x12b7e950UL, 0x8bbeb8eaUL,
  0xfcb9887cUL, 0x62dd1ddfUL, 0x15da2d49UL, 0x8cd37cf3UL, 0xfbd44c65UL, 0x4db26158UL, 0x3ab551ceUL,
  0xa3bc0074UL, 0xd4bb30e2UL, 0x4adfa541UL, 0x3dd895d7UL, 0xa4d1c46dUL, 0xd3d6f4fbUL, 0x4369e96aUL,
  0x346ed9fcUL, 0xad678846UL, 0xda60b8d0UL, 0x44042d73UL, 0x33031de5UL, 0xaa0a4c5fUL, 0xdd0d7cc9UL,
  0x5005713cUL, 0x270241aaUL, 0xbe0b1010UL, 0xc90c2086UL, 0x5768b525UL, 0x206f85b3UL, 0xb966d409UL,
  0xce61e49fUL, 0x5edef90eUL, 0x29d9c998UL, 0xb0d09822UL, 0xc7d7a8b4UL, 0x59b33d17UL, 0x2eb40d81UL,
  0xb7bd5c3bUL, 0xc0ba6cadUL, 0xedb88320UL, 0x9abfb3b6UL, 0x03b6e20cUL, 0x74b1d29aUL, 0xead54739UL,
  0x9dd277afUL, 0x04db2615UL, 0x73dc1683UL, 0xe3630b12UL, 0x94643b84UL, 0x0d6d6a3eUL, 0x7a6a5aa8UL,
  0xe40ecf0bUL, 0x9309ff9dUL, 0x0a00ae27UL, 0x7d079eb1UL, 0xf00f9344UL, 0x8708a3d2UL, 0x1e01f268UL,
  0x6906c2feUL, 0xf762575dUL, 0x806567cbUL, 0x196c3671UL, 0x6e6b06e7UL, 0xfed41b76UL, 0x89d32be0UL,
  0x10da7a5aUL, 0x67dd4accUL, 0xf9b9df6fUL, 0x8ebeeff9UL, 0x17b7be43UL, 0x60b08ed5UL, 0xd6d6a3e8UL,
  0xa1d1937eUL, 0x38d8c2c4UL, 0x4fdff252UL, 0xd1bb67f1UL, 0xa6bc5767UL, 0x3fb506ddUL, 0x48b2364bUL,
  0xd80d2bdaUL, 0xaf0a1b4cUL, 0x36034af6UL, 0x41047a60UL, 0xdf60efc3UL, 0xa867df55UL, 0x316e8eefUL,
  0x4669be79UL, 0xcb61b38cUL, 0xbc66831aUL, 0x256fd2a0UL, 0x5268e236UL, 0xcc0c7795UL, 0xbb0b4703UL,
  0x220216b9UL, 0x5505262fUL, 0xc5ba3bbeUL, 0xb2bd0b28UL, 0x2bb45a92UL, 0x5cb36a04UL, 0xc2d7ffa7UL,
  0xb5d0cf31UL, 0x2cd99e8bUL, 0x5bdeae1dUL, 0x9b64c2b0UL, 0xec63f226UL, 0x756aa39cUL, 0x026d930aUL,
  0x9c0906a9UL, 0xeb0e363fUL, 0x72076785UL, 0x05005713UL, 0x95bf4a82UL, 0xe2b87a14UL, 0x7bb12baeUL,
  0x0cb61b38UL, 0x92d28e9bUL, 0xe5d5be0dUL, 0x7cdcefb7UL, 0x0bdbdf21UL, 0x86d3d2d4UL, 0xf1d4e242UL,
  0x68ddb3f8UL, 0x1fda836eUL, 0x81be16cdUL, 0xf6b9265bUL, 0x6fb077e1UL, 0x18b74777UL, 0x88085ae6UL,
  0xff0f6a70UL, 0x66063bcaUL, 0x11010b5cUL, 0x8f659effUL, 0xf862ae69UL, 0x616bffd3UL, 0x166ccf45UL,
  0xa00ae278UL, 0xd70dd2eeUL, 0x4e048354UL, 0x3903b3c2UL, 0xa7672661UL, 0xd06016f7UL, 0x4969474dUL,
  0x3e6e77dbUL, 0xaed16a4aUL, 0xd9d65adcUL, 0x40df0b66UL, 0x37d83bf0UL, 0xa9bcae53UL, 0xdebb9ec5UL,
  0x47b2cf7fUL, 0x30b5ffe9UL, 0xbdbdf21cUL, 0xcabac28aUL, 0x53b39330UL, 0x24b4a3a6UL, 0xbad03605UL,
  0xcdd70693UL, 0x54de5729UL, 0x23d967bfUL, 0xb3667a2eUL, 0xc4614ab8UL, 0x5d681b02UL, 0x2a6f2b94UL,
  0xb40bbe37UL, 0xc30c8ea1UL, 0x5a05df1bUL, 0x2d02ef8dUL
};
```

3.8.7 Read MD5 File Checksum

This function can be used to read the MD5 chesum of a given file. The checksum will be generated during the request over the actual file data.

File Get MD5 Request

Structure Information: *RCX_FILE_GET_MD5_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	6 + n	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001E68	RCX_FILE_GET_MD5_REQ
Data			
ulChannelNo	UINT32	0 … 3 0xFFFFFFFF	Channel Number Communication Channel System Channel
usFileName Length	UINT16	n	Length of Name Length of the Following File Name (in Bytes)
	UINT8	ASCII	File Name ASCII string, zero terminated

Packet structure reference

```
/* REQUEST MD5 FILE CHECKSUM REQUEST */
#define RCX_FILE_GET_MD5_REQ                0x00001E68

typedef struct RCX_FILE_GET_MD5_REQ_DATA_Ttag
{
  UINT32     ulChannelNo;         /* 0 = Channel 0 ... 3 = Channel 3,         */
                                  /* 0xFFFFFFFF = System, see RCX_FILE_xxxx   */
  UINT16     usFileNameLength;    /* length of NULL-terminated file name      */

  /* a NULL-terminated file name will follow here */
} RCX_FILE_GET_MD5_REQ_DATA_T;

typedef struct RCX_FILE_GET_MD5_REQ_Ttag
{
  PACKET_HEADER                   tHead;       /* packet header              */
  RCX_FILE_GET_MD5_REQ_DATA_T     tData;       /* packet data                */
} RCX_FILE_GET_MD5_REQ_T;
```

File Get MD5 Confirmation

Structure Information: *RCX_FILE_GET_MD5_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	16 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E69	RCX_FILE_GET_MD5_CNF
Data			
abMD5[16]	UINT8	0 … 0xFF	MD5 checksum

Packet structure reference

```
/* REQUEST MD5 FILE CHECKSUM REQUEST */
#define RCX_FILE_GET_MD5_CNF                RCX_FILE_GET_MD5_REQ+1

typedef struct RCX_FILE_GET_MD5_CNF_DATA_Ttag
{
  UINT8       abMD5[16];          /* MD5 checksum                */
} RCX_FILE_GET_MD5_CNF_DATA_T;

typedef struct RCX_FILE_GET_MD5_CNFtag
{
  RCX_PACKET_HEADER             tHead;     /* packet header       */
  RCX_FILE_GET_MD5_CNF_DATA_T   tData;     /* packet data         */
} RCX_FILE_GET_MD5_CNF_T;
```

3.8.8 Read MD5 File Checksum from File Header

System files like the firmware and the configuration database files are containing a MD5 checksum in their file header. Thes checksum can be read by using this function.

File Get Header MD5 Request

Structure Information: *RCX_FILE_GET_HEADER_MD5_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	6+n	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001E72	RCX_FILE_GET_HEADER_MD5_REQ
Data			
ulChannelNo	UINT32	0 … 3 0xFFFFFFFF	Channel Number Communication Channel System Channel
usFileName Length	UINT16	n	Length of Name Length of the Following File Name (in Bytes)
	UINT8	ASCII	File Name ASCII string, zero terminated

Packet structure reference

```
/* REQUEST MD5 FILE HEADER CHECKSUM REQUEST */
#define RCX_FILE_GET_HEADER_MD5_REQ         0x00001E72

/* This packet has the same structure, so we are using a typedef here */
typedef RCX_FILE_GET_MD5_REQ_T  RCX_FILE_GET_HEADER_MD5_REQ_T
```

File Get Header MD5 Confirmation

Structure Information: *RCX_FILE_GET_HEADER_MD5_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	16 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E73	RCX_FILE_GET_HEADER_MD5_CNF
Data			
abMD5[16]	UINT8	0 … 0xFF	MD5 checksum

Packet structure reference

```
/* REQUEST MD5 FILE HEADER CHECKSUM CONFIRMATION */
#define RCX_FILE_GET_HEADER_MD5_CNF         RCX_FILE_GET_HEADER_MD5_REQ+1

/* This packet has the same structure, so we are using a typedef here */
typedef RCX_FILE_GET_MD5_CNF_T  RCX_FILE_GET_HEADER_MD5_CNF_T
```

3.9 Determining the DPM Layout

The layout of the dual-port memory (DPM) can be determined by evaluating the content of the *System Channel Information Block*.

To obtain the logical layout of a channel, the application has to send a packet to the firmware through the system block's mailbox area. The protocol stack replies with one or more messages containing the description of the channel.

Each memory area of a channel has an offset address and an identifier to indicate the type of area (e.g. IO process data image, send/receive mailbox, parameter, status or port specific area.)

DPM Get Block Information Request

Structure Information: *RCX_DPM_GET_BLOCK_INFO_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
`ulDest`	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
`ulLen`	UINT32	8	Packet Data Length (in Bytes)
`ulCmd`	UINT32	0x00001EF8	RCX_DPM_GET_BLOCK_INFO_REQ
Data			
`ulAreaIndex`	UINT32	0 … 7	Area Index (see below)
`ulSubblockIndex`	UINT32	0 … 0xFFFFFFFF	Sub Block Index (see below)

Packet structure reference

```
/* GET BLOCK INFORMATION REQUEST */
#define RCX_DPM_GET_BLOCK_INFO_REQ          0x00001EF8

typedef struct RCX_DPM_GET_BLOCK_INFO_REQ_DATA_Ttag
{
  UINT32  ulAreaIndex;                      /* area index               */
  UINT32  ulSubblockIndex;                  /* sub block index          */
} RCX_DPM_GET_BLOCK_INFO_REQ_DATA_T;

typedef struct RCX_DPM_GET_BLOCK_INFO_REQ_Ttag
{
  RCX_PACKET_HEADER                 tHead;  /* packet header            */
  RCX_DPM_GET_BLOCK_INFO_REQ_DATA_T tData;  /* packet data              */
} RCX_DPM_GET_BLOCK_INFO_REQ_T;
```

Area Index *ulAreaIndex*

This field holds the index of the channel. The system channel is identified by an index number of 0; the handshake has index 1, the first communication channel has index 2 and so on.

Sub Block Index *ulSubblockIndex*

The sub block index field identifies each of the blocks that reside in the dual-port memory interface for the specified communication channel.

DPM Get Block Information Confirmation

The firmware replies with the following message.

Structure Information: *RCX_DPM_GET_BLOCK_INFO_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
`ulLen`	UINT32	28 0	Packet Data Length (in Bytes) If `ulState` = RCX_S_OK Otherwise
`ulState`	UINT32	See Below	Status / Error Code, see Section 6
`ulCmd`	UINT32	0x00001EF9	RCX_GET_BLOCK_INFO_CNF
Data			
`ulAreaIndex`	UINT32	0, 1, … 7	Area Index (Channel Number)
`ulSubblockIndex`	UINT32	0 … 0xFFFFFFFF	Number of Sub Blocks (see below)
`ulType`	UINT32	0 … 0x0009	Type of Sub Block (see below)
`ulOffset`	UINT32	0 … 0xFFFFFFFF	Offset of Sub Block within the Area
`ulSize`	UINT32	0 … 65535	Size of Sub Block (see below)
`usFlags`	UINT16	0 … 0x0023	Transmission Flags of Sub Block (see below)
`usHandshakeMode`	UINT16	0 … 0x0004	Handshake Mode (see below)
`usHandshakeBit`	UINT16	0 … 0x00FF	Bit Position in the Handshake Register
`usReserved`	UINT16	0	Reserved, unused

Packet structure reference

```
/* GET BLOCK INFORMATION CONFIRMATION */
#define RCX_DPM_GET_BLOCK_INFO_CNF           RCX_DPM_GET_BLOCK_INFO_REQ+1

typedef struct RCX_DPM_GET_BLOCK_INFO_CNF_DATA_Ttag
{
  UINT32  ulAreaIndex;      /* area index                                */
  UINT32  ulSubblockIndex;  /* number of sub block                       */
  UINT32  ulType;           /* type of sub block                         */
  UINT32  ulOffset;         /* offset of this sub block within the area  */
  UINT32  ulSize;           /* size of the sub block                     */
  UINT16  usFlags;          /* flags of the sub block                    */
  UINT16  usHandshakeMode;  /* handshake mode                            */
  UINT16  usHandshakeBit;   /* bit position in the handshake register    */
  UINT16  usReserved;       /* reserved                                  */
} RCX_DPM_GET_BLOCK_INFO_CNF_DATA_T;

typedef struct RCX_DPM_GET_BLOCK_INFO_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
  RCX_DPM_GET_BLOCK_INFO_CNF_DATA_T tData;   /* packet data               */
} RCX_DPM_GET_BLOCK_INFO_CNF_T;
```

Area Index *ulAreaIndex*

This field defines the channel number that the block belongs to. The system channel has the number 0; the handshake channel has the number 1; the first communication channel has the number 2 and so on (max. 7).

Sub Block Index *ulSubblockIndex*

This field holds the number of the block.

Sub Block Type *ulType*

This field is used to identify the sub block type. The following types are defined.

Value	Definition / Description
0x0000	UNDEFINED
0x0001	UNKNOWN
0x0002	PROCESS DATA IMAGE
0x0003	HIGH PRIORITY DATA IMAGE
0x0004	MAILBOX
0x0005	COMON CONTROL
0x0006	COMMON STATUS
0x0007	EXTENDED STATUS
0x0008	USER
0x0009	RESERVED
Other values are reserved	

Table 9: Sub Block Type

Offset *ulOffset*

This field holds the offset of the block based on the start offset of the channel.

Size *ulSize*

The size field holds the length of the block section in multiples of bytes.

Transmission Flags *usFlags*

The flags field is separated into nibbles (4 bit entities). The lower nibble is the *Transfer Direction* and holds information regarding the data direction from the view point of the application. The *Transmission Type* nibble defines how data are physically exchanged with this sub block.

Attention: This information is statically set in the firmware during start-up and not updated during run-time even if options are changed by the application (e.g. switch to DMA mode).

Bit No.	Definition / Description
0-3	Transfer Direction 0 UNDEFINED 1 IN (netX to Host System) 2 OUT (Host System to netX) 3 IN – OUT (Bi-Directional) Other values are reserved
4-7	Transmission Type 0 UNDEFINED 1 DPM (Dual-Port Memory) 2 DMA (Direct Memory Access) Other values are reserved
8-15	Reserved, set to 0

Table 10: Transmission Flags

Handshake Mode *usHandshakeMode*

The handshake mode is defined only for IO data images.

Value	Definition / Description
0x0000	UNKNOWN
0x0003	UNCONTROLLED
0x0004	BUFFERED, HOST CONTROLLED
Other values are reserved	

Table 11: Hand Shake Mode

Handshake Bit Position *usHandshakeBit*

Handshake bits are located in the handshake register of a channel and used to synchronise data access to a given data block. The bit position defines the bit number of the used synchronisation bit. The handshake registers itself are located in the *Handshake Channel*. The handling of the handshake cells and synchronisation bit is described in the *netX DPM interface Manual*.

Note: Not all combinations of values from this structure are allowed. Some are even contradictory and do not make sense.

3.10 Security Memory / Flash Device Label

A standard Hilscher device offers a so-called *Security Memory* respectively a *Flash Device Label* to store device specific hardware data.

This memory is divided into 5 zones, a *Configuration Zone* and Zone 0 to Zone 3.

Note: The *Flash Device Label* simulates a *Security Memory* in the Flash of the hardware. It has the same functionality as a *Security Memory*, except it is only available for slave devices and does not provide license information.

Configuration Zone

The *Configuration Zone* holds entries that are predefined by the manufacturer of the *Security Memory* / *Flash Device Label*. This zone is written only during production. It is neither read nor writable.

The zone includes serial number, device number, hardware revision, production date, device class and hardware compatibility.

The information is shown in the system information block of the DPM and is part of the packet which is described in section 3.3.1 “Read Hardware” on page 15.

Zone 0
Is encrypted and contains netX related hardware features and license information.
Zone 0 is not read or writable by an application.

Zone 1
Is used for general hardware configuration settings like Ethernet MAC address and SDRAM timing parameter.
Zone 1 is read and writeable by an application.

Zone 2
Is used for PCI configuration and operating system depending parameters.
Zone 2 is read and writeable by an application.

Zone 3
Is under control of a user application running on the netX to store its data.
Zone 3 is read and writeable by an application.

Note: Usually it is not necessary to write to zones 1 or 2 nor is it recommended. Changes can cause memory access faults, configuration or communication problems!

Zones 1 and 2 of the Security Memory are protected by a checksum (see page 64 for details).

3.10.1 Security Memory Zones Content

Zone 1 – Hardware Configuration

Attention: **Please read chapter *3.7 MAC Address Handling***.
Because a netX device will **occupy up to 4 MAC addresses**, even if only one address can be stored.

Offset	Type	Name	Description
0x00	UINT8	`MacAddress[6]`	Ethernet Medium Access Address, 6 Bytes
0x06	UINT32	`SdramGeneralCtrl`	SDRAM control register value
0x0A	UINT32	`SdramTimingCtrl`	SDRAM timing register value
0x0E	UINT8	`SdramSizeExp`	SDRAM size in Mbytes
0x0F	UINT16	`HwOptions[4]`	Hardware Assembly Option, 4 Words
0x17	UINT8	`BootOption`	Boot Option
0x18	UINT8	`Reserved[6]`	Reserved, 6 Bytes
0x1E	UINT8	`Zone1Revision`	Revision Structure of Zone 1
0x1F	UINT8	`Zone1Checksum`	Checksum of Byte 0 to 30

Table 12: Hardware Configuration (Zone 1)

Zone 2 – PCI System and OS Settings

Offset	Type	Name	Description
0x00	UINT16	`PciVendorID`	PCI Settings
0x02	UINT16	`PciDeviceID`	
0x04	UINT8	`PciSubClassCode`	
0x05	UINT8	`PciClassCode`	
0x06	UINT16	`PciSubsystemVendorID`	
0x08	UINT16	`PciSubsystemDeviceID`	
0x0A	UINT8	`PciSizeTarget[3]`	
0x0D	UINT8	`PciSizeIO`	
0x0E	UINT8	`PciSizeROM[3]`	
0x11	UINT8	`Reserved`	
0x12	UINT8	`OsSettings[12]`	OS Related Information, 12 Bytes
0x1E	UINT8	`Zone2Revision`	Revision Structure of Zone 2
0x1F	UINT8	`Zone2Checksum`	Checksum of Byte 0 to 30

Table 13: PCI System and OS Setting (Zone 2)

Zone 3 – User Specific Zone

Offset	Type	Name	Description
0 – 0x1F	UINT8	`UserSpecific[32]`	Reserved, 32 Byte

Table 14: User Specific Zone (Zone 3)

Memory Zones Structure Reference

```
typedef struct RCX_SECURITY_MEMORY_ZONE1tag
{
  UINT8   MacAddress[6];          /* Ethernet medium access address      */
  UINT32  SdramGeneralCtrl;       /* SDRAM control register value        */
  UINT32  SdramTimingCtrl;        /* SDRAM timing register value         */
  UINT8   SdramSizeExp;           /* SDRAM size in Mbytes                */
  UINT16  HwOptions[4];           /* hardware assembly option            */
  UINT8   BootOption;             /* boot option                         */
  UINT8   Reserved[6];            /* reserved (6 bytes)                  */
  UINT8   Zone1Revision;          /* revision structure of zone 1        */
  UINT8   Zone1Checksum;          /* checksum of byte 0 to 30            */
} RCX_SECURITY_MEMORY_ZONE1;

typedef struct RCX_SECURITY_MEMORY_ZONE2tag
{
  UINT16  PciVendorID;            /* PCI settings                        */
  UINT16  PciDeviceID;            /* PCI settings                        */
  UINT8   PciSubClassCode;        /* PCI settings                        */
  UINT8   PciClassCode;           /* PCI settings                        */
  UINT16  PciSubsystemVendorID;   /* PCI settings                        */
  UINT16  PciSubsystemDeviceID;   /* PCI settings                        */
  UINT8   PciSizeTarget[3];       /* PCI settings                        */
  UINT8   PciSizeIO;              /* PCI settings                        */
  UINT8   PciSizeROM[3];          /* PCI settings                        */
  UINT8   Reserved;
  UINT8   OsSettings[12];         /* OS Related Information              */
  UINT8   Zone2Revision;          /* Revision Structure of Zone 2        */
  UINT8   Zone2Checksum;          /* Checksum of Byte 0 to 30            */
} RCX_SECURITY_MEMORY_ZONE2;

typedef struct RCX_SECURITY_MEMORY_ZONE3tag
{
  UINT8   UserSpecific[32];       /* user specific area                  */
} RCX_SECURITY_MEMORY_ZONE3;
```

3.10.2 Checksum

Zones 0, 1 and 2 of the Security Memory are protected by a checksum.

The netX operating system provides functions that automatically calculate the checksum when zones 1 and 2 are written. So in a packet to write these zones the checksum field is set to zero. The packet to read these zones returns the checksum stored in the Security Memory.

3.10.3 Zone Read

Security Memory Read Request

Read information from the Security Memory (if available).

Structure Information: *RCX_SECURITY_EEPROM_READ_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001EBC	RCX_SECURITY_EEPROM_READ_REQ
Data			
ulZoneId	UINT32	0x00000001 0x00000002 0x00000003	Zone Identifier RCX_SECURITY_EEPROM_ZONE_1 RCX_SECURITY_EEPROM_ZONE_2 RCX_SECURITY_EEPROM_ZONE_3

Packet structure reference

```
/* READ SECURITY EEPROM REQUEST */
#define RCX_SECURITY_EEPROM_READ_REQ         0x00001EBC

/* Memory Zones */
#define RCX_SECURITY_EEPROM_ZONE_1           0x00000001
#define RCX_SECURITY_EEPROM_ZONE_2           0x00000002
#define RCX_SECURITY_EEPROM_ZONE_3           0x00000003

typedef struct RCX_SECURITY_EEPROM_READ_REQ_DATA_Ttag
{
  UINT32  ulZoneId;                      /* zone identifier             */
} RCX_SECURITY_EEPROM_READ_REQ_DATA_T;

typedef struct RCX_SECURITY_EEPROM_READ_REQ_Ttag
{
  RCX_PACKET_HEADER                    tHead;     /* packet header       */
  RCX_SECURITY_EEPROM_READ_REQ_DATA_T  tData;     /* packet data         */
} RCX_SECURITY_EEPROM_READ_REQ_T;
```

Security Memory Read Confirmation

Structure Information: *RCX_SECURITY_EEPROM_READ_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	 32 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EBD	RCX_SECURITY_EEPROM_READ_CNF
Data			
abZoneData[32]	UINT8	0 … 0xFF	Data from requested zone (size is 32 Byte)

Packet structure reference

```
/* READ SECURITY EEPROM CONFIRMATION */
#define RCX_SECURITY_EEPROM_READ_CNF         RCX_SECURITY_EEPROM_READ_REQ+1

typedef struct RCX_SECURITY_EEPROM_READ_CNF_DATA_Ttag
{
  UINT8   abZoneData[32];                    /* zone data               */
} RCX_SECURITY_EEPROM_READ_CNF_DATA_T;

typedef struct RCX_SECURITY_EEPROM_READ_CNF_Ttag
{
  RCX_PACKET_HEADER                   tHead;  /* packet header           */
  RCX_SECURITY_EEPROM_READ_CNF_DATA_T tData;  /* packet data             */
} RCX_SECURITY_EEPROM_READ_CNF_T;
```

3.10.4 Zone Write

Note: To avoid changing essential parameters in the security memory by accident, the application must read the entire zone first, modify fields as required and write the entire zone afterwards.
Changing parameter like SDRAM register or PCI settings may cause unwanted behavior of the netX chip and it might get into a state where no further operation is possible.

Security Memory Write Request

Structure Information: *RCX_SECURITY_EEPROM_WRITE_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	36	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001EBE	RCX_SECURITY_EEPROM_WRITE_REQ
Data			
ulZoneId	UINT32	0x00000001 0x00000002 0x00000003	Zone Identifier RCX_SECURITY_EEPROM_ZONE_1 RCX_SECURITY_EEPROM_ZONE_2 RCX_SECURITY_EEPROM_ZONE_3
abZoneData[32]	UINT8	0 ... 0xFF	Zone data (size is 32 byte)

Packet structure reference

```
/* WRITE SECURITY EEPROM REQUEST */
#define RCX_SECURITY_EEPROM_WRITE_REQ        0x00001EBE

/* Memory Zones */
#define RCX_SECURITY_EEPROM_ZONE_1           0x00000001
#define RCX_SECURITY_EEPROM_ZONE_2           0x00000002
#define RCX_SECURITY_EEPROM_ZONE_3           0x00000003

typedef struct RCX_SECURITY_EEPROM_WRITE_REQ_DATA_Ttag
{
  UINT32   ulZoneId;          /* zone ID, see RCX_SECURITY_EEPROM_ZONE_x   */
  UINT8    abZoneData[32];    /* zone data                                 */
} RCX_SECURITY_EEPROM_WRITE_REQ_DATA_T;

typedef struct RCX_SECURITY_EEPROM_WRITE_REQ_Ttag
{
  RCX_PACKET_HEADER                     tHead;   /* packet header         */
  RCX_SECURITY_EEPROM_WRITE_REQ_DATA_T  tData;   /* packet data           */
} RCX_SECURITY_EEPROM_WRITE_REQ_T;
```

Note: The configuration zone and zone 0 are neither readable nor writable.

Security Memory Write Confirmation

Structure Information: *RCX_SECURITY_EEPROM_WRITE_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EBF	RCX_SECURITY_EEPROM_WRITE_CNF

Packet structure reference

```
/* WRITE SECURITY EEPROM CONFIRMATION */
#define RCX_SECURITY_EEPROM_WRITE_CNF        RCX_SECURITY_EEPROM_WRITE_REQ+1

typedef struct RCX_SECURITY_EEPROM_WRITE_CNF_Ttag
{
  RCX_PACKET_HEADER       tHead;          /* packet header               */
} RCX_SECURITY_EEPROM_WRITE_CNF_T;
```

3.11 License Information

HW Read License Request

The application uses the following packet in order to obtain license information from the netX firmware. The packet is send through the system mailbox.

Structure Information: *RCX_HW_LICENSE_INFO_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulCmd	UINT32	0x00001EF4	RCX_HW_LICENSE_INFO_REQ

Packet structure reference

```
/* OBTAIN LICENSE INFORMATION REQUEST */
#define RCX_HW_LICENSE_INFO_REQ              0x00001EF4

typedef struct RCX_HW_LICENSE_INFO_REQ_Ttag
{
  RCX_PACKET_HEADER     tHead;            /* packet header               */
} RCX_HW_LICENSE_INFO_REQ_T;
```

HW Read License Confirmation

Structure Information: *RCX_HW_LICENSE_INFO_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	12 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EF5	RCX_HW_LICENSE_INFO_CNF
Data			
ulLicenseFlags1	UINT32	0 … 0xFFFFFFFF	License Flags 1
ulLicenseFlags2	UINT32	0 … 0xFFFFFFFF	License Flags 2
usNetxLicenseID	UINT16	0 … 0xFFFF	netX License Identification
usNetxLicenseFlags	UINT16	0 … 0xFFFF	netX License Flags

Packet structure reference

```
/* OBTAIN LICENSE INFORMATION CONFIRMATION */
#define RCX_HW_LICENSE_INFO_CNF              RCX_HW_LICENSE_INFO_REQ+1

typedef struct RCX_HW_LICENSE_INFO_CNF_DATA_Ttag
{
  UINT32   ulLicenseFlags1;        /* License Flags 1        */
  UINT32   ulLicenseFlags2;        /* License Flags 2        */
  UINT16   usNetxLicenseID;        /* License ID             */
  UINT16   usNetxLicenseFlags;     /* License Flags          */
} RCX_HW_LICENSE_INFO_CNF_DATA_T;

typedef struct RCX_HW_LICENSE_INFO_CNFtag
{
  RCX_PACKET_HEADER                tHead;  /* packet header              */
  RCX_HW_LICENSE_INFO_CNF_DATA_T   tData;  /* packet data                */
} RCX_HW_LICENSE_INFO_CNF_T;
```

3.12 System Performance Counter

The netX firmware offers several performance counters allowing an application to evalute the CPU usage of the netX system.

Get Perf Counters Request

This packet is used to performance counters from a netX fimware.

Structure Information: *RCX_GET_PERF_COUNTERS_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001ED4	RCX_GET_PERF_COUNTERS_REQ
Data			
usStartToken	UINT16	0 … 0xFFFF	Start token of values
usTokenCount	UINT16	0 … 0xFFFF	Number of tokens

Packet structure reference

```
/* READ PERFORMANCE COUNTER REQUEST */
#define RCX_GET_PERF_COUNTERS_REQ            0x00001ED4

typedef struct RCX_GET_PERF_COUNTERS_REQ_DATA_Ttag
{
  UINT16 usStartToken;
  UINT16 usTokenCount;
} RCX_GET_PERF_COUNTERS_REQ_DATA_T;

typedef struct RCX_GET_PERF_COUNTERS_REQ_Ttag
{
  RCX_PACKET_HEADER                      tHead;   /* packet header      */
  RCX_GET_PERF_COUNTERS_REQ_DATA_T       tData;   /* packet data        */
} RCX_GET_PERF_COUNTERS_REQ_T;
```

Get Perf Counters Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_GET_PERF_COUNTERS_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	4 + 8 + (n * 8) 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001ED5	RCX_GET_PERF_COUNTERS_CNF
Data			
usStartToken	UINT16	from Request	Start token given in the request
usTokenCount	UINT16	n	Max. number of token in the following array
tPerfSystemUptime	Structure		System up time
atPerfCounters[1]	Structure		Counters

Packet structure reference

```
/* READ PERFORMANCE COUNTER CONFIRMATION */
#define RCX_GET_PERF_COUNTERS_CNF           RCX_GET_PERF_COUNTERS_REQ+1

typedef struct RCX_PERF_COUNTER_DATA_Ttag
{
  UINT32 ulNanosecondsLower;
  UINT32 ulNanosecondsUpper;
} RCX_PERF_COUNTER_DATA_T;

typedef struct RCX_GET_PERF_COUNTERS_CNF_DATA_Ttag
{
  UINT16                   usStartToken;
  UINT16                   usTokenCount;
  RCX_PERF_COUNTER_DATA_T  tPerfSystemUptime;
/*  dynamic array, length is given indirectly by ulLen                    */
  RCX_PERF_COUNTER_DATA_T  atPerfCounters[1];
} RCX_GET_PERF_COUNTERS_CNF_DATA_T;

typedef struct RCX_GET_PERF_COUNTERS_CNF_Ttag
{
  RCX_PACKET_HEADER                       tHead;   /* packet header      */
  RCX_GET_PERF_COUNTERS_CNF_DATA_T        tData;   /* packet data        */
} RCX_GET_PERF_COUNTERS_CNF_T;
```

3.13 Real-Time Clock

The netX hardware may support a real time clock. If present, the following commands can be used to set and get the time from the system.

After power cycling, the time is set to a predefined value if the clock has no auxiliary power supply (backup battery, gold cap…etc.).

Time Command Request

The time command can be used to set the clock and to read the time or the status of the clock. The packet is send through the system mailbox.

Structure Information: *RCX_TIME_CMD_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	12	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001ED8	RCX_TIME_COMMAND_REQ
Data			
ulTimeCmd	UINT32	0x00000001 0x00000002 0x00000003	Time Command TIME_CMD_GETSTATE TIME_CMD_GETTIME TIME_CMD_SETTIME
ulData	UINT32	0 0 Time in Seconds	Data (Content corresponds to command) TIME_CMD_GETSTATE (see below) TIME_CMD_GETTIME (see below) TIME_CMD_SETTIME (see below)
ulReserved	UINT32	0	Reserved, set to 0

Packet structure reference

```
/* Time Packet Command */
#define RCX_TIME_COMMAND_REQ              0x00001ED8

/* Time Commands */
#define TIME_CMD_GETSTATE                 0x00000001   /* get state      */
#define TIME_CMD_GETTIME                  0x00000002   /* get time       */
#define TIME_CMD_SETTIME                  0x00000003   /* set time       */

typedef struct RCX_TIME_CMD_DATA_Ttag
{
  UINT32  ulTimeCmd;                      /* time command                */
  UINT32  ulData;                         /* data, corresponds to command */
  UINT32  ulReserved;                     /* Reserved                    */
} RCX_TIME_CMD_DATA_T;

typedef struct RCX_TIME_CMD_REQ_Ttag
{
  RCX_PACKET_HEADER       tHead;          /* packet header               */
  RCX_TIME_CMD_DATA_T     tData;          /* packet data                 */
} RCX_TIME_CMD_REQ_T;
```

Time Command Field *ulTimeCmd*

The time command field holds the sub function in the time command.

Value	Definition / Description
0x00000001	TIME_CMD_GETSTATE returns the current status of the clock function
0x00000002	TIME_CMD_GETTIME returns the current time from the clock
0x00000003	TIME_CMD_SETTIME allows setting the time
Other values are reserved.	

Table 15: Time Command Field

Data Field *ulData* – Set Time

For the Set Time command, the data field holds the time in seconds since January, 1 1970 / 00:00:00 (midnight).

Otherwise this field is set to 0 (zero).

Time Command Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_TIME_CMD_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	12 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001ED9	RCX_TIME_COMMAND_CNF
Data			
ulTimeCmd	UINT32	0x00000001 0x00000002 0x00000003	Time Commands TIME_CMD_GETSTATE TIME_CMD_GETTIME TIME_CMD_SETTIME
ulData	UINT32	Clock Status Time in Seconds Time in Seconds	Data content corresponds to command TIME_CMD_GETSTATE (see below) TIME_CMD_GETTIME (see below) TIME_CMD_SETTIME (see below)
ulReserved	UINT32	0	Reserved, set to 0

Packet structure reference

```
/* Time Packet Command */
#define RCX_TIME_COMMAND_CNF              RCX_TIME_COMMAND_REQ+1

typedef struct RCX_TIME_CMD_DATA_Ttag
{
  UINT32   ulTimeCmd;                     /* time command                */
  UINT32   ulData;                        /* corresponds to command      */
  UINT32   ulReserved;                    /* reserved                    */
} RCX_TIME_CMD_DATA_T;

typedef struct RCX_TIME_CMD_CNF_Ttag
{
  RCX_PACKET_HEADER       tHead;          /* packet header               */
  RCX_TIME_CMD_DATA_T     tData;          /* packet data                 */
} RCX_TIME_CMD_CNF_T;
```

TIME_CMD_GETSTATE

For the *TIME_CMD_GETSTATE* command, the following bit field information is returned *ulData.*

31	30	…	12	11	10	9	8	7	6	5	4	3	2	1	0	*ulData*
														Clock Type 00 = No RTC 01 = RTC internal 10 = RTC external 11 = RTC emulated		
													Clock Status 0 = Time not valid 1 = Time valid			
Unused, set to zero																

Table 16: Clock Status

Bit No.	Definition / Description
0-1	Clock Type 0 = No RTC — Unknown RTC or driver not initialized 1 = RTC internal — netX internal RTC using 32.768 kHz clock 2 = RTC external — External RTC (PCF8563) connected via I2C 3 = RTC emulated — No RTC hardware present, use system tick
2	Clock Status 0 = Time not valid — Time was not set, RTC not initialized, battery failure, etc. 1 = Time valid — Clock was initialized and time was set
Other values are reserved.	

Table 17: Clock Status

TIME_CMD_GETTIME

The *TIME_CMD_GETTIME* command returns the actual system time in *ulData.*

The time is given in format: **seconds since January, 1 1970 / 00:00:00** (midnight).

TIME_CMD_SETTIME

TIME_CMD_SETTIME command will set the clock to the time given in *ulData*. The confirmation will always return the data from the request.

3.14 Start RAM based Firmware on netX

The following packet is used to start (or instantiate for that matter) a firmware on netX when this firmware is executed from RAM. RAM based firmware must be downloaded to the hardware before it can started.

If the netX firmware is executed from Flash, this packet has no effect.

Start Firmware Request

The application uses the following packet in order to start a firmware that is executed from RAM. The packet is send through the system mailbox to the netX firmware. The channel number *ulChannelNo* has to be set to identify the firmware file in a channel.

Structure Information: *RCX_CHANNEL_INSTANTIATE_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001EC4	RCX_CHANNEL_INSTANTIATE_REQ
Data			
ulChannelNo	UINT32	 0x00000000 0x00000001 0x00000002 0x00000003	Channel Number RCX_COMM_CHANNEL_0 RCX_COMM_CHANNEL_1 RCX_COMM_CHANNEL_2 RCX_COMM_CHANNEL_3

Packet structure reference

```
/* INSTANTIATE FIRMWARE REQUEST */
#define RCX_CHANNEL_INSTANTIATE_REQ         0x00001EC4

/* Channel Number */
#define RCX_COMM_CHANNEL_0                  0x00000000
#define RCX_COMM_CHANNEL_1                  0x00000001
#define RCX_COMM_CHANNEL_2                  0x00000002
#define RCX_COMM_CHANNEL_3                  0x00000003

typedef struct RCX_CHANNEL_INSTANTIATE_REQ_DATA_Ttag
{
  UINT32                        ulChannelNo;         /* channel number    */
} RCX_CHANNEL_INSTANTIATE_REQ_DATA_T;

typedef struct RCX_CHANNEL_INSTANTIATE_REQ_Ttag
{
  RCX_PACKET_HEADER                     tHead;       /* packet header     */
  RCX_CHANNEL_INSTANTIATE_REQ_DATA_T    tData;       /* packet data       */
} RCX_CHANNEL_INSTANTIATE_REQ_T;
```

Start Firmware Confirmation

The system channel returns the following packet.

Structure Information: *RCX_CHANNEL_INSTANTIATE_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EC5	RCX_CHANNEL_INSTANTIATE_CNF

Packet structure reference

```
/* INSTANTIATE FIRMWARE CONFIRMATION */
#define RCX_CHANNEL_INSTANTIATE_CNF         RCX_CHANNEL_INSTANTIATE_REQ+1

typedef struct RCX_CHANNEL_INSTANTIATE_CNF_Ttag
{
  RCX_PACKET_HEADER               tHead;            /* packet header     */
} RCX_CHANNEL_INSTANTIATE_CNF_T;
```

3.15 Second Stage Bootloader

The following services are only available if the 2nd Stage Bootloader is active.

3.15.1 Format the Default Partition

This function can be used to format the system partition of the target file system.

Attention: Formating the partition will erase all files in the file system

Format Request

Structure Information: *RCX_FORMAT_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
ulLen	UINT32	8	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001ED6	RCX_FORMAT_REQ
Data			
ulFlags	UINT32	0x00000000 0x00000001	Type of format operation RCX_FORMAT_REQ_DATA_FLAGS_QUICKFORMAT RCX_FORMAT_REQ_DATA_FLAGS_FULLFORMAT
ulReserved	UINT32	0	Reserved, unsed

Packet structure reference

```
/* FORMAT REQUEST */
#define RCX_FORMAT_REQ                      0x00001ED6

#define RCX_FORMAT_REQ_DATA_FLAGS_QUICKFORMAT 0x00000000
#define RCX_FORMAT_REQ_DATA_FLAGS_FULLFORMAT  0x00000001

typedef struct RCX_FORMAT_REQ_DATA_Ttag
{
  UINT32  ulFlags;
  UINT32  ulReserved;
} RCX_FORMAT_REQ_DATA_T;

typedef struct RCX_FORMAT_REQ_Ttag
{
  RCX_PACKET_HEADER                     tHead;                  /* packet header */
  RCX_FORMAT_REQ_DATA_T                 tData;
} RCX_FORMAT_REQ_T;
```

Format Confirmation

Structure Information: *RCX_FORMAT_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	8 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001ED7	RCX_FORMAT_CNF
Data			
ulExtended ErrorInfo	UINT32	0x00000001 != 0xFF	Set if full format failed during an erase or verify operation. IO_TYPE_SPI (if failed during erase operation). Full format is not supported for IO_TYPE_RAM and IO_TYPE_SDMMC. Last byte verified at offset *ulErrorOffset* (if failed during verify operation).
ulErrorOffset	UINT32		Offset the error was encountered on

Packet structure reference

```
/* FORMAT CONFIRMATION */
#define RCX_FORMAT_CNF                    RCX_FORMAT_REQ+1

typedef struct RCX_FORMAT_CNF_DATA_Ttag
{
  /* Valid if format has failed during a full format with an error during
     erase / verify (ulSta = TLR_E_RCX_FORMAT_ERASE_FAILED or
     TLR_E_RCX_FORMAT_VERIFY_FAILED */
  UINT32  ulExtendedErrorInfo;
  UINT32  ulErrorOffset;
} RCX_FORMAT_CNF_DATA_T;

typedef struct RCX_FORMAT_CNF_Ttag
{
  RCX_PACKET_HEADER                     tHead;                  /* packet header */
  RCX_FORMAT_CNF_DATA_T                 tData;
} RCX_FORMAT_CNF_T;
```

4 Communication Channel services

The following functions corresponding to information and functionalities of the so-called communication channels.

4.1 Function overview

Communication Channel services		
Service	**Command definition**	**Page**
Communication Channel Information Blocks		
Read the *Common Control Block* of a channel	RCX_CONTROL_BLOCK_REQ	80
Read the *Common Status Block* of a channel	RCX_DPM_GET_COMMON_STATE_REQ	82
Read the *Extended Status Block* of a channel	RCX_DPM_GET_EXTENDED_STATE_REQ	84
Read Communication Flag States		
Read the communication flags of a specified communication channel	RCX_DPM_GET_COMFLAG_INFO_REQ	86
Read the I/O Process Data Image Size		
Read the configured size of the I/O process data image	RCX_GET_DPM_IO_INFO_REQ	88
Channel Initialization		
Re-initialize / re-configure a protocol stack	RCX_CHANNEL_INIT_REQ	90
Delete Protocol Stack Configuration		
Delete a actual configuration of a protocol stack	RCX_DELETE_CONFIG_REQ	91
Lock / Unlock Configuration		
Lock or unlock a configuration against changes	RCX_LOCK_UNLOCK_CONFIG_REQ	92
Start / Stop Communication		
Start or stop network communication	RCX_START_STOP_COMM_REQ	93
Channel Watchdog Time		
Read the actual watchdog time of a communication channel	RCX_GET_WATCHDOG_TIME_REQ	94
Set the watchdog time of a communication channel	RCX_SET_WATCHDOG_TIME_REQ	95

4.2 Communication Channel Information Blocks

The following packets are used to make certain data blocks, located in the communication channel, available for read access through the communication channel mailbox.

These data blocks are useful for applications and configuration tool like SYCON.net because the blocks contain important states and infromation about a fieldbus protocol stack.

If the requested data block exceeds the maximum mailbox size, the block is transferred in a sequenced or fragmented manner (see *netX Dual-Port Memory Interface Manual* for more information on *Packet Fragmentation*).

4.2.1 Read Common Control Block

Note: A detailed description of the *Common Status Block* can be found in the *netX DPM Interface Manual.*

Read Common Control Block Request

This packet is used to request the *Common Control Block*. The firmware ignores the channel Identifier `ulChannelId`, if the packet is passed through the channel mailbox.

Structure Information: *RCX_READ_COMM_CNTRL_BLOCK_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
`ulDest`	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	UINT32	4	Packet Data Length (in Bytes)
`ulCmd`	UINT32	0x00001E3A	RCX_CONTROL_BLOCK_REQ
Data			
`ulChannelId`	UINT32	0 … 7	Channel Identifier Port Number, Channel Number

Packet structure reference

```
/* READ COMMUNICATION CONTROL BLOCK REQUEST */
#define RCX_CONTROL_BLOCK_REQ                0x00001E3A

typedef struct RCX_READ_COMM_CNTRL_BLOCK_REQ_DATA_Ttag
{
  UINT32   ulChannelId;                        /* channel identifier      */
} RCX_READ_COMM_CNTRL_BLOCK_REQ_DATA_T;

typedef struct RCX_READ_COMM_CNTRL_BLOCK_REQ_Ttag
{
  RCX_PACKET_HEADER                    tHead;     /* packet header        */
  RCX_READ_COMM_CNTRL_BLOCK_REQ_DATA_T tData;     /* packet data          */
} RCX_READ_COMM_CNTRL_BLOCK_REQ_T;
```

Read Common Control Block Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_READ_COMM_CNTRL_BLOCK_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	8 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001E3B	RCX_CONTROL_BLOCK_CNF
Data			
tControl	Structure		Communication Control Block

Packet structure reference

```
/* READ COMMUNICATION CONTROL BLOCK CONFIRMATION */
#define RCX_CONTROL_BLOCK_CNF                RCX_CONTROL_BLOCK_REQ+1

typedef struct RCX_READ_COMM_CNTRL_BLOCK_CNF_DATA_Ttag
{
  NETX_CONTROL_BLOCK                    tControl;  /* control block       */
} RCX_READ_COMM_CNTRL_BLOCK_CNF_DATA_T;

typedef struct RCX_READ_COMM_CNTRL_BLOCK_CNF_Ttag
{
  RCX_PACKET_HEADER                     tHead;     /* packet header       */
  RCX_READ_COMM_CNTRL_BLOCK_CNF_DATA_T  tData;     /* packet data         */
} RCX_READ_COMM_CNTRL_BLOCK_CNF_T;
```

4.2.2 Read Common Status Block

The *Common Status Block* contains common fieldbus information offered by all fieldbus systems.

Note: A detailed description of the *Common Status Block* can be found in the *netX DPM interface Manual.*

Read Common Status Block Request

This packet is used to request the *Common Status Block*.

The firmware ignores the channel identifier *ulChannelId*, if the packet is passed through the channel mailbox.

Structure Information: *RCX_READ_COMMON_STS_BLOCK_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001EFC	RCX_DPM_GET_COMMON_STATE_REQ
Data			
ulChannelId	UINT32	0 … 7	Channel Identifier Port Number, Channel Number

Packet structure reference

```
/* READ COMMON STATUS BLOCK REQUEST */
#define RCX_DPM_GET_COMMON_STATE_REQ         0x00001EFC

typedef struct RCX_READ_COMMON_STS_BLOCK_REQ_DATA_Ttag
{
  UINT32 ulChannelId;                          /* channel identifier       */
} RCX_READ_COMMON_STS_BLOCK_REQ_DATA_T;

typedef struct RCX_READ_COMMON_STS_BLOCK_REQ_Ttag
{
  RCX_PACKET_HEADER                     tHead;     /* packet header        */
  RCX_READ_COMMON_STS_BLOCK_REQ_DATA_T  tData;     /* packet data          */
} RCX_READ_COMMON_STS_BLOCK_REQ_T;
```

Read Common Status Block Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_READ_COMMON_STS_BLOCK_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
`ulLen`	UINT32	64 0	Packet Data Length (in Bytes) If `ulState` = `RCX_S_OK` Otherwise
`ulState`	UINT32	See Below	Status / Error Code, see Section 6
`ulCmd`	UINT32	0x00001EFD	RCX_DPM_GET_COMMON_STATE_CNF
Data			
`tCommonStatus`	Structure		Common Status Block

Packet structure reference

```
/* READ COMMON STATUS BLOCK CONFIRMATION */
#define RCX_DPM_GET_COMMON_STATE_CNF         RCX_DPM_GET_COMMON_STATE_REQ+1

typedef struct RCX_READ_COMMON_STS_BLOCK_CNF_DATA_Ttag
{
  NETX_COMMON_STATUS_BLOCK              tCommonStatus;  /* common status   */
} RCX_READ_COMMON_STS_BLOCK_CNF_DATA_T;

typedef struct RCX_READ_COMMON_STS_BLOCK_CNF_Ttag
{
  RCX_PACKET_HEADER                     tHead;  /* packet header          */
  RCX_READ_COMMON_STS_BLOCK_CNF_DATA_T  tData;  /* packet data            */
} RCX_READ_COMMON_STS_BLOCK_CNF_T;
```

4.2.3 Read Extended Status Block

This packet is used to read the *Extended Status Block*. This block contains protocol stack and fieldbus specific information (e.g. specific master state information).

Note: A detailed description of the *Extended Status Block* can be found in the *netX DPM interface Manual.*

Read Extended Status Block Request

The firmware ignores the channel identifier *ulChannelId*, if the packet is passed through the channel mailbox.

Structure Information: *RCX_DPM_GET_EXTENDED_STATE_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	12	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00001EFE	RCX_DPM_GET_EXTENDED_STATE_REQ
Data			
ulOffset	UINT32	0 … 431	Byte offset in extended status block structure
ulDataLen	UINT32	1 … 432	Length in byte read
ulChannel Index	UINT32	0 … 7	Channel Identifier Port Number, Channel Number

Packet structure reference

```
/* READ EXTENDED STATUS BLOCK REQUEST */
#define RCX_DPM_GET_EXTENDED_STATE_REQ       0x00001EFE

typedef struct RCX_DPM_GET_EXTENDED_STATE_REQ_DATA_Ttag
{
  UINT32 ulOffset;             /* offset in extended status block        */
  UINT32 ulDataLen;            /* size of block to read                  */
  UINT32 ulChannelIndex;       /* channel number                         */
} RCX_DPM_GET_EXTENDED_STATE_REQ_DATA_T;

typedef struct RCX_DPM_GET_EXTENDED_STATE_REQ_Ttag
{
  RCX_PACKET_HEADER                     tHead;   /* packet header       */
  RCX_DPM_GET_EXTENDED_STATE_REQ_DATA_T tData;   /* packet data         */
} RCX_DPM_GET_EXTENDED_STATE_REQ_T;
```

Read Extended Status Block Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_DPM_GET_EXTENDED_STATE_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	1 … 432 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EFF	RCX_DPM_GET_EXTENDED_STATE_CNF
ulExt	UINT32	0x00000000 0x00000080 0x000000C0 0x00000040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
Data			
ulOffset	UINT32	0 … 431	Byte offset in extended status block structure
ulDataLen	UINT32	1 … 432	Length in byte
abData[432]	UINT8	0 … n	Extended Status Block data

Packet structure reference

```
/* READ EXTENDED STATUS BLOCK CONFIRMATION */
#define RCX_DPM_GET_EXTENDED_STATE_CNF      RCX_DPM_GET_EXTENDED_STATE_REQ+1

typedef struct RCX_DPM_GET_EXTENDED_STATE_CNF_DATA_Ttag
{
  UINT32 ulOffset;             /* offset in extended status block        */
  UINT32 ulDataLen;            /* size of block returned                 */
  UINT8  abData[432];          /* data block                             */
} RCX_DPM_GET_EXTENDED_STATE_CNF_DATA_T;

typedef struct RCX_DPM_GET_EXTENDED_STATE_CNF_Ttag
{
  RCX_PACKET_HEADER                     tHead;     /* packet header       */
  RCX_DPM_GET_EXTENDED_STATE_CNF_DATA_T tData;     /* packet data         */
} RCX_DPM_GET_EXTENDED_STATE_CNF_T;
```

4.3 Read the Communication Flag States

This service allows reading the *Communication Flags* of a specified channel. These flags are used to synchronise the data transfer between a host and a netX target and containing general system states information like *NCF_COMMUNICATING* or *NCF_ERROR*.

Note: The functionality and the content of the *Communication Flags* are described in the *netX DPM Interface Manual.*

DPM Get ComFlag Info Request

Structure Information: *RCX_DPM_GET_COMFLAG_INFO_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
`ulDest`	UINT32	0x00000000	RCX_PACKET_DEST_SYSTEM
`ulLen`	UINT32	4	Packet Data Length (in Bytes)
`ulCmd`	UINT32	0x00001EFA	RCX_DPM_GET_COMFLAG_INFO_REQ
Data			
`ulAreaIndex`	UINT32	0 … 7	Area Index (see below)

Packet structure reference

```
/* DPM GET COMFLAG INFO REQUEST */
#define RCX_DPM_GET_COMFLAG_INFO_REQ       0x00001EFA

typedef struct RCX_DPM_GET_COMFLAG_INFO_REQ_DATA_Ttag
{
  UINT32                            ulAreaIndex;            /* area index */
} RCX_DPM_GET_COMFLAG_INFO_REQ_DATA_T;

typedef struct RCX_DPM_GET_COMFLAG_INFO_REQ_Ttag
{
  RCX_PACKET_HEADER_T                   tHead;              /* packet header */
  RCX_DPM_GET_COMFLAG_INFO_REQ_DATA_T   tData;              /* packet data */
} RCX_DPM_GET_COMFLAG_INFO_REQ_T;
```

Area Index: *ulAreaIndex*

This field holds the index of the channel. The area index counts all channels in a firmware starting with index 0 for the system channel. The first communication channel will have the index 2 and so on.

Area Index Tabel

Index	Channel Description
0	System Channel
1	Handshake Channel
2	Communication Channel 0
3	Communication Channel 1
4	Communication Channel 2
5	Communication Channel 3

DPM Get ComFlag Info Confirmation

Structure Information: *RCX_DPM_GET_COMFLAG_INFO_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	12 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00001EFB	RCX_DPM_GET_COMFLAG_INFO_CNF
Data			
ulAreaIndex	UINT32	0, 1, … 7	Area Index (see above)
ulNetxComFlag	UINT32	Bit Field	Current netX Communication Flags
ulHostComFlag	UINT32	Bit Field	Current Host Communication Flags

Packet structure reference

```
/* DPM GET COMFLAG INFO CONFIRMATION */
#define RCX_DPM_GET_COMFLAG_INFO_CNF       RCX_DPM_GET_COMFLAG_INFO_REQ+1

typedef struct RCX_DPM_GET_COMFLAG_INFO_CNF_DATA_Ttag
{
  UINT32  ulAreaIndex;                         /* area index */
  UINT32  ulNetxComFlag;
  UINT32  ulHostComFlag;
} RCX_DPM_GET_COMFLAG_INFO_CNF_DATA_T;

typedef struct RCX_DPM_GET_COMFLAG_INFO_CNF_Ttag
{
  RCX_PACKET_HEADER_T                 tHead;   /* packet header */
  RCX_DPM_GET_COMFLAG_INFO_CNF_DATA_T tData;   /* packet data */
} RCX_DPM_GET_COMFLAG_INFO_CNF_T;
```

4.4 Read I/O Process Data Image Size

The application can request information about the length of the configured I/O process data image. The length information is useful to adjust copy functions in terms of the amount of data which are defined by the fieldbus protocol configuration.

Note: Some of the protocol stacks are able to map additional state informations into the I/O data image.
This addtional length must be obained from the extended state block information (see *4.2.3 Read Extended Status Block*).

The answer packet returns the offset of the first used byte used in the I/O data image and the length of configured I/O data.

Get DPM I/O Information Request

This packet is used to obtain offset and length of the used I/O data space.

Structure Information: *RCX_GET_DPM_IO_INFO_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	UINT32	0x00002F0C	RCX_GET_DPM_IO_INFO_REQ

Packet structure reference

```
/* GET DPM I/O INFORMATION REQUEST */
#define RCX_GET_DPM_IO_INFO_REQ              0x00002F0C

typedef struct RCX_GET_DPM_IO_INFO_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header     */
} RCX_GET_DPM_IO_INFO_REQ_T;
```

Get DPM I/O Information Confirmation

The confirmation packet returns offset and length of the requested input and the output data area. The application may receive the packet in a sequenced manner. So the *ulExt* field has to be evaluated!

Structure Information: *RCX_GET_DPM_IO_INFO_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	4+(20 * n) 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulId	UINT32	From Request	Packet Identification as Unique Number
ulState	UINT32	See Below	Status / Error Code see Section 6
ulCmd	UINT32	0x00002F0D	RCX_GET_DPM_IO_INFO_CNF
ulExt	UINT32	0x00000000 0x00000080 0x000000C0 0x00000040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
Structure Information			
ulNumIOBlock Info	UINT32	0 … 10	Number **n** of Block Definitions Below
atIoBlockInfo [2]	Array of Structure		I/O Block Definition Structure(s) RCX_DPM_IO_BLOCK_INFO

Packet structure reference

```
/* GET DPM I/O INFORMATION CONFIRMATION */
#define RCX_GET_DPM_IO_INFO_CNF             RCX_GET_DPM_IO_INFO_REQ+1

typedef struct RCX_DPM_IO_BLOCK_INFO_Ttag
{
  UINT32  ulSubblockIndex;  /* index of sub block                      */
  UINT32  ulType;           /* type of sub block                       */
  UINT16  usFlags;          /* flags of the sub block                  */
  UINT16  usReserved;       /* reserved                                */
  UINT32  ulOffset;         /* offset                                  */
  UINT32  ulLength;         /* length of I/O data in bytes             */
} RCX_DPM_IO_BLOCK_INFO_T;

typedef struct RCX_GET_DPM_IO_INFO_CNF_DATA_Ttag
{
  UINT32                   ulNumIOBlockInfo; /* Number of IO Block Info       */
  RCX_DPM_IO_BLOCK_INFO_T  atIoBlock[2];     /* Array of I/O Block information */
} RCX_GET_DPM_IO_INFO_CNF_DATA_T;

typedef struct RCX_GET_DPM_IO_INFO_CNF_Ttag
{
  RCX_PACKET_HEADER               tHead;   /* packet header             */
  RCX_GET_DPM_IO_INFO_CNF_DATA_T  tData;   /* packet data               */
} RCX_GET_DPM_IO_INFO_CNF_T;
```

4.5 Channel Initialization

A *Channel Initialization* affects only the designated communication channel. It forces the protocol stack to immediately close all network connections and to proceed with a re-initialization. While the stack is started the configuration settings are evaluated again.

NOTE If the configuration is locked, re-initialization of a channel is not allowed.

Channel Initialization Request

The packet is send through the channel mailbox.

Structure Information: *RCX_CHANNEL_INIT_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	UINT32	0x00002F80	RCX_CHANNEL_INIT_REQ

Packet structure reference

```
/* CHANNEL INITIALIZATION REQUEST */
#define RCX_CHANNEL_INIT_REQ                0x00002F80

typedef struct RCX_CHANNEL_INIT_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
} RCX_CHANNEL_INIT_REQ_T;
```

Channel Initialization Confirmation

The channel firmware returns the following packet.

Structure Information: *RCX_CHANNEL_INIT_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F81	RCX_CHANNEL_INIT_CNF

Packet structure reference

```
/* CHANNEL INITIALIZATION CONFIRMATION */
#define RCX_CHANNEL_INIT_CNF                RCX_CHANNEL_INIT_REQ+1

typedef struct RCX_CHANNEL_INIT_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
} RCX_CHANNEL_INIT_CNF_T;
```

4.6 Delete Protocol Stack Configuration

A protocol stack can be configured via a configuration database file or via packet services (*Set Configuration Packets*).

Note: This service has no effect, if the protocol stack is configured via a configuration database file. To delete a configuration file, the standard file functions has to be used (see page 50 for details).

If configured via packets, the configuration settings are stored by the protocol stack in RAM. The configuration will be lost on a channel reset or power cycle.

However, the configuration cannot be deleted, as long as the *Configuration Locked* flag in *ulCommunicationCOS* is set.

After a channel initialization, the protocol stack won't startup properly due to the missing configuration.

The following packet is used to delete the configuration from RAM.

Delete Configuration Request

The application uses the following packet in order to delete the current configuration of the protocol stack. The packet is send through the channel mailbox to the protocol stack.

Structure Information: *RCX_DELETE_CONFIG_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	UINT32	0x00002F14	RCX_DELETE_CONFIG_REQ

Packet structure reference

```
/* DELETE CONFIGURATION REQUEST */
#define RCX_DELETE_CONFIG_REQ                0x00002F14

typedef struct RCX_DELETE_CONFIG_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;    /* packet header             */
} RCX_DELETE_CONFIG_REQ_T;
```

Delete Configuration Confirmation

The system returns the following packet.

Structure Information: *RCX_DELETE_CONFIG_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F15	RCX_DELETE_CONFIG_CNF

Packet structure reference

```
/* DELETE CONFIGURATION CONFIRMATION */
#define RCX_DELETE_CONFIG_CNF                RCX_DELETE_CONFIG_REQ+1

typedef struct RCX_DELETE_CONFIG_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;    /* packet header             */
} RCX_DELETE_CONFIG_CNF_T;
```

4.7 Lock / Unlock Configuration

The lock configuration mechanism is used to prevent the configuration settings from being altered during protocol stack execution. The request packet is passed through the channel mailbox only and also affects the *Configuration Locked* flag in the *Common Control Block.*

The protocol stack modifies this flag in order to signal its current state.

Lock / Unlock Config Request

The packet is send through the channel mailbox.

Structure Information: *RCX_LOCK_UNLOCK_CONFIG_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F32	RCX_LOCK_UNLOCK_CONFIG_REQ
Data			
ulParam	UINT32	0x00000001 0x00000002	Parameter Lock Configuration Unlock Configuration

Packet structure reference

```
/* LOCK – UNLOCK CONFIGURATION REQUEST */
#define RCX_LOCK_UNLOCK_CONFIG_REQ          0x00002F32

typedef struct RCX_LOCK_UNLOCK_CONFIG_REQ_DATA_Ttag
{
  UINT32  ulParam;                            /* lock/unlock parameter  */
} RCX_LOCK_UNLOCK_CONFIG_REQ_DATA_T;

typedef struct RCX_LOCK_UNLOCK_CONFIG_REQ_Ttag
{
  RCX_PACKET_HEADER                  tHead;   /* packet header          */
  RCX_LOCK_UNLOCK_CONFIG_REQ_DATA_T  tData;   /* packet data            */
} RCX_LOCK_UNLOCK_CONFIG_REQ_T;
```

Lock / Unlock Config Confirmation

The channel firmware returns the following packet.

Structure Information: *RCX_LOCK_UNLOCK_CONFIG_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F33	RCX_LOCK_UNLOCK_CONFIG_CNF

Packet structure reference

```
/* LOCK – UNLOCK CONFIGURATION CONFIRMATION */
#define RCX_LOCK_UNLOCK_CONFIG_CNF          RCX_LOCK_UNLOCK_CONFIG_REQ+1

typedef struct RCX_LOCK_UNLOCK_CONFIG_CNF_Ttag
{
  RCX_PACKET_HEADER    tHead;            /* packet header               */
} RCX_LOCK_UNLOCK_CONFIG_CNF_T;
```

4.8 Start / Stop Communication

The command is used to force a protocol stack to start or stop network communication. It is passed to the protocol stack through the channel mailbox. Starting and stopping network communication affects the *Bus On* flag (see *Communication Change of State* register).

Start / Stop Communication Request

The application uses the following packet in order to start or stop network communication. The packet is send through the channel mailbox.

Structure Information: *RCX_START_STOP_COMM_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F30	RCX_START_STOP_COMM_REQ
Data			
ulParam	UINT32	0x00000001 0x00000002	Parameter RCX_COMM_START_CMD RCX_COMM_STOP_CMD

Packet structure reference

```
/* START – STOP COMMUNICATION REQUEST */
#define RCX_START_STOP_COMM_REQ              0x00002F30

#define RCX_COMM_START_CMD 0x00000001
#define RCX_COMM_STOP_CMD  0x00000002

typedef struct RCX_START_STOP_COMM_REQ_DATA_Ttag
{
  UINT32  ulParam;                      /* start/stop communication      */
} RCX_START_STOP_COMM_REQ_DATA_T;

typedef struct RCX_START_STOP_COMM_REQ_Ttag
{
  RCX_PACKET_HEADER               tHead;       /* packet header          */
  RCX_START_STOP_COMM_REQ_DATA_T  tData;       /* packet data            */
} RCX_START_STOP_COMM_REQ_T;
```

Start / Stop Communication Confirmation

The firmware returns the following packet.

Structure Information: *RCX_START_STOP_COMM_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F31	RCX_START_STOP_COMM_CNF

Packet structure reference

```
/* START – STOP COMMUNICATION CONFIRMATION    */
#define RCX_START_STOP_COMM_CNF              RCX_START_STOP_COMM_REQ+1

typedef struct RCX_START_STOP_COMM_CNF_Ttag
{
  RCX_PACKET_HEADER    tHead;            /* packet header               */
} RCX_START_STOP_COMM_CNF_T;
```

4.9 Channel Watchdog Time

The communication channel watchdog time can be retrieved and set using the following watchdog time commands.

4.9.1 Get Channel Watchdog Time

Get Watchdog Time Request

The application can use the following packet to read the actual configured watchdog.

Structure Information: *RCX_GET_WATCHDOG_TIME_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	UINT32	0x00002F02	RCX_GET_WATCHDOG_TIME_REQ

Packet structure reference

```
/* GET WATCHDOG TIME REQUEST */
#define RCX_GET_WATCHDOG_TIME_REQ           0x00002F02

typedef struct RCX_GET_WATCHDOG_TIME_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header     */
} RCX_GET_WATCHDOG_TIME_REQ_T;
```

Get Watchdog Time Confirmation

The system channel returns the following packet.

Structure Information: *RCX_GET_WATCHDOG_TIME_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	4 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F03	RCX_GET_WATCHDOG_TIME_CNF
Data			
ulWdgTime	UINT32	0 20 … 0xFFFF	Watchdog Time in milliseconds [ms] = not set 20 > WDT < 0xFFFF

Packet structure reference

```
/* GET WATCHDOG TIME CONFIRMATION */
#define RCX_GET_WATCHDOG_TIME_CNF           RCX_GET_WATCHDOG_TIME_REQ+1

typedef struct RCX_GET_WATCHDOG_TIME_CNF_DATA_Ttag
{
  UINT32               ulWdgTime;   /* current watchdog time            */
} RCX_GET_WATCHDOG_TIME_CNF_DATA_T;

typedef struct RCX_GET_WATCHDOG_TIME_CNF_Ttag
{
  RCX_PACKET_HEADER                  tHead;     /* packet header          */
  RCX_GET_WATCHDOG_TIME_CNF_DATA_T   tData;     /* packet data            */
} RCX_GET_WATCHDOG_TIME_CNF_T;
```

4.9.2 Set Watchdog Time

The application can use the following packet to set the watchdog time of a *Communication Channel*.

Set Watchdog Time Request

Structure Information: *RCX_SET_WATCHDOG_TIME_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F04	RCX_SET_WATCHDOG_TIME_REQ
Data			
ulWdgTime	UINT32	0 20 … 65535	Watchdog Time Watchdog inactive Watchdog time in milliseconds

Packet structure reference

```
/* SET WATCHDOG TIME REQUEST */
#define RCX_SET_WATCHDOG_TIME_REQ           0x00002F04

typedef struct RCX_SET_WATCHDOG_TIME_REQ_DATA_Ttag
{
  /** watchdog time in milliseconds */
  UINT32 ulWdgTime;
} RCX_SET_WATCHDOG_TIME_REQ_DATA_T;

typedef struct RCX_SET_WATCHDOG_TIME_REQ_Ttag
{
  RCX_PACKET_HEADER                 tHead;
  RCX_SET_WATCHDOG_TIME_REQ_DATA_T  tData;
} RCX_SET_WATCHDOG_TIME_REQ_T;
```

Set Watchdog Time Confirmation

The system channel returns the following packet.

Structure Information: *RCX_SET_WATCHDOG_TIME_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F05	RCX_SET_WATCHDOG_TIME_CNF

Packet structure reference

```
/* SET WATCHDOG TIME CONFIRMATION */
#define RCX_SET_WATCHDOG_TIME_CNF    RCX_SET_WATCHDOG_TIME_REQ+1

typedef struct RCX_SET_WATCHDOG_TIME_CNF_Ttag
{
  RCX_PACKET_HEADER                 tHead;   /* packet header             */
} RCX_SET_WATCHDOG_TIME_CNF_T;
```

5 Protocol Stack services

Protocol stack services are functions handled by the protocol stacks.

These functions are also fieldbus depending and not all of the fieldbus systems are offering the same information or functions.

5.1 Function overview

Protocol stack services		
Service	**Command definition**	**Page**
Change the Process Data Handshake Configuration		
Set the mode how I/O data are synchronized with the host	RCX_SET_HANDSHAKE_CONFIG_REQ	97
Modify Configuration Settings		
Set protocol stack configuration parameters to new values	RCX_SET_FW_PARAMETER_REQ	102
Network Connection State		
Obtain a list of slave which are configured, active or faulted	RCX_GET_SLAVE_HANDLE_REQ	106
Obtain a slave connection information	RCX_GET_SLAVE_CONN_INFO_REQ	108
Protocol Stack Notifications / Indications		
Register an application to be able to receive notifications from a protocol stack	RCX_REGISTER_APP_REQ	111
Unregister an application from receiving notifications	RCX_UNREGISTER_APP_REQ	112
Link Status Changed Service		
Activate a link status change notification	RCX_LINK_STATUS_CHANGE_IND	113
Perform a Bus Scan		
Scan for available devices on the fieldbus devices	RCX_BUSSCAN_REQ	115
Get Information about a Fieldbus Device		
Read the fieldbus depending information of a device	RCX_GET_DEVICE_INFO_REQ	117
Configuration in Run		
Verify a modified configuration database file	RCX_VERIFY_DATABASE_REQ	119
Activate the modified configuration	RCX_ACTIVATE_DATABASE_REQ	121

5.2 Set Handshake Configuration

The application uses the following packet in order to set the process data handshake mode.

Note: Handshake modes are described in the *netX DPM Interface Manual*. It is also protocol stack depending which handshake modes are supported.

Note: Changing the handshake mode by the application is only allowed before I/O data is exchanged with the protocol stack. Changing the mode during I/O data exchanges with the protocol stack could lead into unpredictable states in the I/O synchronisation.

Set Handshake Configuration request

The packet is send through the channel mailbox to the protocol stack.

Variable	Type	Value / Range	Description
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	20	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F34	RCX_SET_HANDSHAKE_CONFIG_REQ
Data			
bPDInHskMode	UINT8		Input process data handshake mode
bPDInSource	UINT8	0	Input process data trigger source; unused, set to zero
usPDInErrorTh	UINT16		Threshold for input process data handshake handling errors
bPDOutHskMode	UINT8		Output process data handshake mode
bPDOutSource	UINT8	0	Output process data trigger source; unused, set to zero
usPDOutErrorTh	UINT16	0 … 0xFFFF	Threshold for output process data handshake handling errors
bSyncHskMode	UINT8		Synchronization handshake mode
bSyncSource	UINT8		Synchronization source
usSyncErrorTh	UINT16	0 … 0xFFFF	Threshold for synchronization handshake handling errors
aulReserved[2]	UINT32	0	Reserved for future use; set to zero

Table 18: RCX_SET_HANDSHAKE_CONFIG_REQ_T - Set Handshake Configuration

Packet structure reference

```
/* SET HANDSHAKE CONFIGURATION REQUEST */
#define RCX_SET_HANDSHAKE_CONFIG_REQ        0x00002F34

typedef struct RCX_SET_HANDSHAKE_CONFIG_REQ_DATA_Ttag
{
  UINT8   bPDInHskMode;   /* input process data handshake mode          */
  UINT8   bPDInSource;    /* input process data trigger source          */
  UINT16  usPDInErrorTh;  /* threshold for input data handshake handling errors    */
  UINT8   bPDOutHskMode;  /* output process data handshake mode         */
  UINT8   bPDOutSource;   /* output process data trigger source         */
  UINT16  usPDOutErrorTh; /* threshold for output data handshake handling errors   */
  UINT8   bSyncHskMode;   /* synchronization handshake mode             */
  UINT8   bSyncSource;    /* synchronization source                     */
  UINT16  usSyncErrorTh;  /* threshold for sync handshake handling errors*/
  UINT32  aulReserved[2]; /* reserved for future use                    */
} RCX_SET_HANDSHAKE_CONFIG_REQ_DATA_T;
```

```
typedef struct RCX_SET_HANDSHAKE_CONFIG_REQ_Ttag
{
  RCX_PACKET_HEADER                       tHead;   /* packet header      */
  RCX_SET_HANDSHAKE_CONFIG_REQ_DATA_T     tData;   /* packet data        */
} RCX_SET_HANDSHAKE_CONFIG_REQ_T;
```

Set Handshake Configuration Confirmation

The following packet is returned by the firmware.

Structure Information: *RCX_SET_HANDSHAKE_CONFIG_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F35	RCX_SET_HANDSHAKE_CONFIG_CNF

Packet structure reference

```
/* SET HANDSHAKE CONFIGURATION CONFIRMATION */
#define RCX_SET_HANDSHAKE_CONFIG_CNF         RCX_SET_HANDSHAKE_CONFIG_REQ+1

typedef struct RCX_SET_HANDSHAKE_CONFIG_CNF_Ttag
{
  RCX_PACKET_HEADER                       tHead;   /* packet header      */
} RCX_SET_HANDSHAKE_CONFIG_CNF_T;
```

5.3 Modify Configuration Settings

The *Modify Configuration Settings* functionality allows to selectively changing configuration parameters or settings of a slave protocol stacks which is already configured by a configuration database file (e.g. config.nxd).

The subsequent modification of configuration settings is particularly useful if the same configuration database file is used for a number of identical slave devices where each of the devices needs some individual settings like a unique network address or station name.

Note: Modifying configuration settings is only possible if the protocol stack is configured by a configuration database file (e.g. config.nxd) and the network startup behavior, given by the configuration database, is set to ***Controlled Start of Communication***.

Example of parameters which usually have to be modified:

- Station / Network Address
- Baudrate
- Name of Station (PROFINET Device only)
- Device Identification (EtherCAT Slave only)
- Second Station Address (EtherCAT Slave only)

General Configuration Handling

In general, a protocol stack can be configured in 3 different ways.

- SYCON.net configuration database file
- iniBatch database file (via netX Configuration Tool)
- Configuration via *Set Configuration Request* packets

After power-on reset, a protocol stack first checks if a configuration database file (e.g. config.nxd) is available. If so, the configuration will be evaluated and no other configuration will be accepted from this point (see *Set Configuration* packets). In case a configuration database file could not be found, the firmware checks next if an *iniBatch* database file is available and if so, it proceeds in the same way. If none of the two database files are available, the protocol stack will remain in unconfigured state and waits until an application starts to send configuration packets to the stack.

To be able to use the modification service, the protocol stack must be in a specific state. It must be configured by a configuration database file and the network startup behaviour in the configuration database must be set to *Controlled Start of Communication.* Only in this state, where the protocol stack waits on a BUS-ON command, he will accept modification commands.

Flowchart

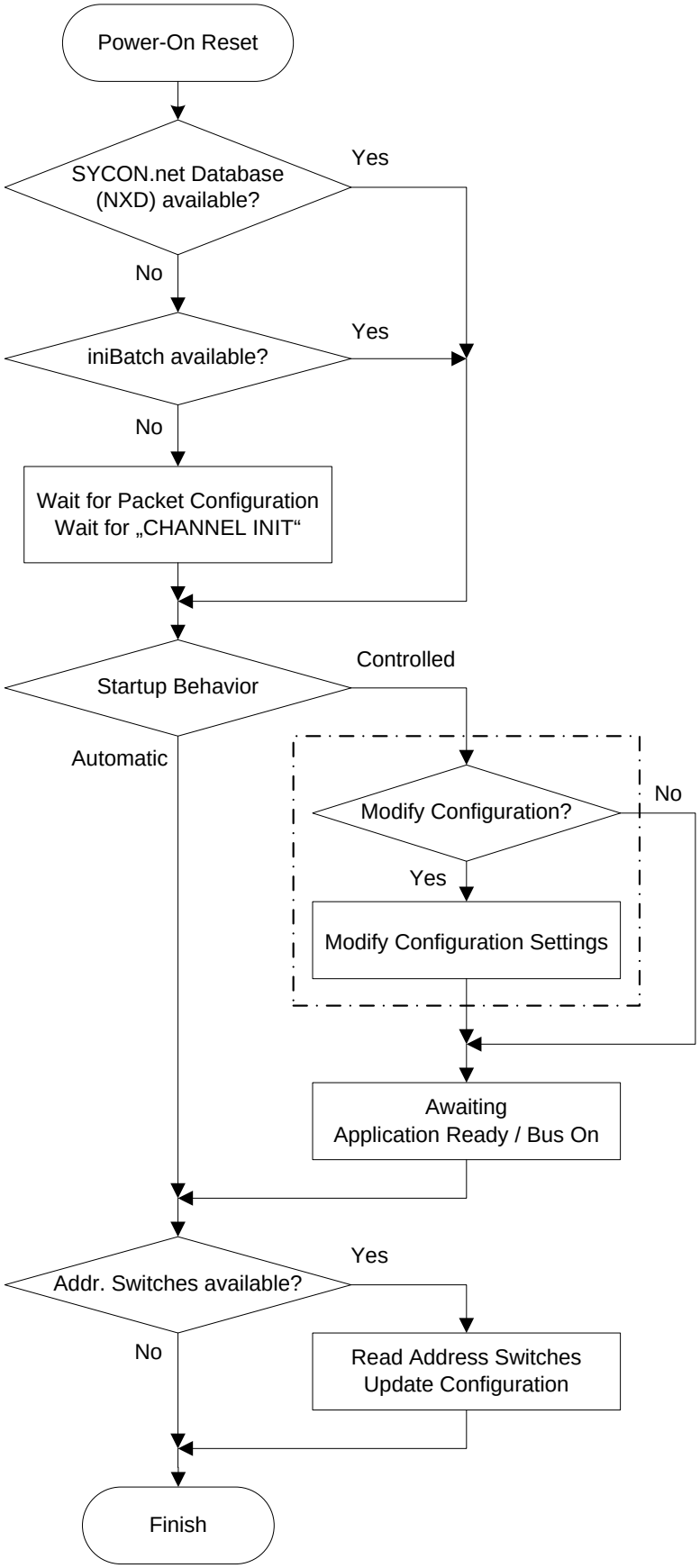

Figure 3: Flowchart Modify Configuration Settings

Behavior when configuration is locked

The protocol stack returns no error code when the host application tries to modify the configuration settings while the configuration is locked (see section 4.7 Lock / Unlock Configuration on page 92).

Behavior while network communication / bus on is set

The protocol stack returns no error code when the host application tries to modify the configuration settings during network communication or if BUS_ON is set. The new parameter value is not applied to the current configuration. This behavior is necessary because some fieldbus systems are required to react when certain configuration parameters change during runtime.

For example, the DeviceNet firmware shall indicate an error status via its LED if a new network address was assigned during runtime.

Note: During network communication, the *Get Parameter* command can be used to read the currently used parameter.

Behavior during channel initialization

During channel initialization (see *netX Dual-Port Memory Interface Manual* for more details) all parameters set by the *Set Parameter* command are discarded and the original from the configuration database are used again.

5.3.1 Set Parameter Data

This service allows a host application to modify certain protocol stack parameters from the current configuration. This requires that *Controlled Start of Communication* is set in the configuration database file and the protocol stack is waiting for the *BUS ON / APPLICATION READY* command.

Set Parameter Request

Depending on the stack implementation the service allows to set one or more parameters in one request. Please consult the protocol stack manual which parameters are changeable. The packet is send through the channel mailbox.

Variable	Type	Value / Range	Description
`ulDest`	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	UINT32	8 + n	Packet Data Length (in Bytes)
`ulCmd`	UINT32	0x00002F86	RCX_SET_FW_PARAMETER_REQ
`ulParameterID`	UINT32	0 … 0xFFFFFFFF	Parameter identifier, see Table 20 and Table 21
`ulParameterLen`	UINT32	n	Length of abParameter in byte
`abParameter[4]`	UINT8	m	Parameter value, byte array

Table 19: RCX_SET_FW_PARAMETER_REQ_T – Set Parameter Data

Packet structure reference

```
/* SET FIRMWARE PARAMETER REQUEST */
#define RCX_SET_FW_PARAMETER_REQ           0x00002F86

typedef struct RCX_SET_FW_PARAMETER_REQ_DATA_Ttag
{
  UINT32  ulParameterID;                   /* parameter identifier       */
  UINT32  ulParameterLen;                  /* parameter length           */
  UINT8   abParameter[4];                  /* parameter                  */
} RCX_SET_FW_PARAMETER_REQ_DATA_Ttag;

typedef struct RCX_SET_FW_PARAMETER_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header              */
  RCX_SET_FW_PARAMETER_REQ_DATA_T  tData;   /* packet data                */
} RCX_SET_FW_PARAMETER_REQ_T;
```

Parameter Identifier *ulParameterID*

The Parameter Identifier is encoded as outlined below (0xPCCCCNNN).

31	…	28	27	26	25	…	14	13	12	11	10	…	2	1	0	
										NNN = unique number						
			CCCC = protocol class (see *usProtocolClass* in the *netX Dual-Port Memory Interface Manual*)													
P = prefix (always 0x3)																

Table 20: Encoding Parameter Identifier

The following parameter identifiers are defined.

Name	Code	Type	Size	Description of Parameter
PID_STATION_ADDRESS	0x30000001	UINT32	4 Byte	Station Address
PID_BAUDRATE	0x30000002	UINT32	4 Byte	Baud Rate
PID_PN_NAME_OF_STATION	0x30015001	UINT8	240 Byte	PROFINET: Name of Station
PID_ECS_DEVICE_IDENTIFICATION	0x30009001	UINT16	4 Byte	EtherCAT: Value for Explicit Device Identification
PID_ECS_SCND_STATION_ADDRESS	0x30009002	UINT16	4 Byte	EtherCAT: Second Station Address
All other codes are reserved for future use.				

Table 21: Defined Parameter Identifier

Set Parameter Confirmation

The following packet describes the answer of the *Set Parameter Request.*

Structure Information: *RCX_SET_FW_PARAMETER_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F87	RCX_SET_FW_PARAMETER_CNF

Packet structure reference

```
/* SET FIRMWARE PARAMETER CONFIRMATION */
#define RCX_SET_FW_PARAMETER_CNF           RCX_SET_FW_PARAMETER_REQ+1

typedef struct RCX_SET_FW_PARAMETER_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header            */
} RCX_SET_FW_PARAMETER_CNF_T;
```

5.4 Network Connection State

This section explains how an application can obtain connection status information about slave devices from a master protocol stack. Hence the packets below are only supported by master protocol stacks. Slave stacks do not support this function and will rejects the request with an error code.

5.4.1 Mechanism

The application can request information about the status of network slaves in regards of their cyclic connection (Non-cyclic connections are not handled in here).

The protocol stack returns a list of handles where each handle represents one slave device.

Note: A handle of a slave is not its MAC ID, station or node address nor an IP address.

The following lists are available.

- **List of Configured Slaves**
  This list represents all network nodes that are configured via a configuration database file or via packet services.
- **List of Actived Slaves**
  This list holds network nodes that are configured (see above) and actively communicating to the network master.

  Note: This is not a 'Life List'! The list contains only nodes included in the configuration.
- **List of Faulted Slaves**
  This list contains handles of all configured nodes that currently encounter some sort of connection problem (e.g. disconnected, hardware or configuration problems).

Handling procedure

At first an application has to send a *Get Slave Handle Request* to obtain the list of slaves.

Note: Handles may change after reconfiguration or power-on reset.

With the handles returned by *Get Slave Handle Request*, the application can use the *Get Slave Connection Information Request* to read the slave's current network status.

The network status information is always fieldbus specific and to be able to evaluate the slave information data, the returned information also contains the unique identification number `ulStructID`. By using `ulStructID` the application is able to identify the delivered data structured.

Identification numbers and structures are described in the corresponding protocol stack interface manual and coressponding structure definitions can be found in the protocol specif header files.

In a flawless network (all configured slaves are working properly) the list of configured slaves is identical to the list of activated slaves and both list containing the same handles. In case of a slave failure, the corresponding slave handle will be removed from the active slave list and moved to the faulty slave list while the list of configured slaves remains always constant.

If an application want to check, if the fieldbus system (all slaves) workings correctly, it has to compare the *List of Configured Slaves* against the *List of Active Slaves*. If both lists are identical, all slaves are active on the bus.

Faulty slaves are always shown in the *List of Faulted Slaves* which contains the corresponding slave handle. Depending on the fieldbus system a faulty slave may or may not appear in the *List of Active Slaves*.

The reason why slaves are not working correctly could differ between fieldbus systems. Obvious causes are:

- Inconsistent configuration between master and slave
- Slave parameter data faults
- Disconnected network cable

Note: Diagnostic functionalities and diagnostic information details are heavily depending on the fieldbus system. Therefore only the handling to get the information is specified. The data evaluation must be done by the application using the fieldbus specific documentations and definitions.

5.4.2 Obtain List of Slave Handles

Get Slave Handle Request

The host application uses the packet below in order to request a list of slaves depending on the requested type:

- *List of Configured Slaves* (*RCX_LIST_CONF_SLAVES*)
- *List of Activated Slaves* (*RCX_LIST_ACTV_SLAVES*)
- *List of Faulted Slaves* (*RCX_LIST_FAULTED_SLAVES*)

Structure Information: *RCX_PACKET_GET_SLAVE_HANDLE_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F08	RCX_GET_SLAVE_HANDLE_REQ
Data			
ulParam	UINT32	Parameter 0x00000001 0x00000002 0x00000003	Parameter RCX_LIST_CONF_SLAVES RCX_LIST_ACTV_SLAVES RCX_LIST_FAULTED_SLAVES

Packet structure reference

```
/* GET SLAVE HANDLE REQUEST */
#define RCX_GET_SLAVE_HANDLE_REQ             0x00002F08

/* LIST OF SLAVES */
#define RCX_LIST_CONF_SLAVES                 0x00000001
#define RCX_LIST_ACTV_SLAVES                 0x00000002
#define RCX_LIST_FAULTED_SLAVES              0x00000003

typedef struct RCX_PACKET_GET_SLAVE_HANDLE_REQ_DATA_Ttag
{
  UINT32  ulParam;                          /* type of list               */
} RCX_PACKET_GET_SLAVE_HANDLE_REQ_DATA_T;

typedef struct RCX_PACKET_GET_SLAVE_HANDLE_REQ_Ttag
{
  RCX_PACKET_HEADER                       tHead;    /* packet header     */
  RCX_PACKET_GET_SLAVE_HANDLE_REQ_DATA_T  tData;    /* packet data       */
} RCX_PACKET_GET_SLAVE_HANDLE_REQ_T;
```

Get Slave Handle Confirmation

This is the answer to the *RCX_GET_SLAVE_HANDLE_REQ* command. The answer packet contains a list of slave handles. Each handle in the returned list describes a slave device where the slave state corresponds to the requested list type (*configured*, *activated* or *faulted*).

Structure Information: *RCX_PACKET_GET_SLAVE_HANDLE_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
`ulLen`	UINT32	4 * (1 + n) 0	Packet Data Length (in Bytes) If `ulState` = RCX_S_OK Otherwise
`ulState`	UINT32	See Below	Status / Error Code, see Section 6
`ulCmd`	UINT32	0x00002F09	RCX_GET_SLAVE_HANDLE_CNF
Data			
`ulParam`	UINT32	0x00000001 0x00000002 0x00000003	Parameter RCX_LIST_CONF_SLAVES RCX_LIST_ACTV_SLAVES RCX_LIST_FAULTED_SLAVES
`aulHandle[1]`	UINT32	0 … 0xFFFFFFFF	Slave Handle, Number of Handles is **n**

Packet structure reference

```
/* GET SLAVE HANDLE CONFIRMATION */
#define RCX_GET_SLAVE_HANDLE_CNF             RCX_GET_SLAVE_HANDLE_REQ+1

typedef struct RCX_PACKET_GET_SLAVE_HANDLE_CNF_DATA_Ttag
{
  UINT32  ulParam;                          /* type of list               */
  /* list of handles follows here                                        */
  UINT32  aulHandle[1];
} RCX_PACKET_GET_SLAVE_HANDLE_CNF_DATA_T

typedef struct RCX_PACKET_GET_SLAVE_HANDLE_CNF_Ttag
{
  RCX_PACKET_HEADER                        tHead;     /* packer header     */
  RCX_PACKET_GET_SLAVE_HANDLE_CNF_DATA_T  tData;     /* packet data       */
} RCX_PACKET_GET_SLAVE_HANDLE_CNF_T;
```

5.4.3 Obtain Slave Connection Information

Get Slave Connection Information Request

Using the handles from section 5.4.2, the application can request network status information for each of the configured network slaves.

Structure Information: *RCX_PACKET_GET_SLAVE_CONN_INFO_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F0A	RCX_GET_SLAVE_CONN_INFO_REQ
Data			
ulHandle	UINT32	0 … 0xFFFFFFFF	Slave Handle

Packet structure reference

```
/* SLAVE CONNECTION INFORMATION REQUEST */
#define RCX_GET_SLAVE_CONN_INFO_REQ          0x00002F0A

typedef struct RCX_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_Ttag
{
  UINT32  ulHandle;                                    /* slave handle  */
} RCX_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_T;

typedef struct RCX_PACKET_GET_SLAVE_CONN_INFO_REQ_Ttag
{
  RCX_PACKET_HEADER                              tHead;   /* packer header */
  RCX_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_T    tData;   /* packet data   */
} RCX_PACKET_GET_SLAVE_CONN_INFO_REQ_T;
```

Get Slave Connection Information Confirmation

The confimation contains the fieldbus specific state information of the requested slave defined in *ulHandle*.

The identification number *ulStructID* defines the fieldbus specific information data structure following the *ulStructID* element in the packet.

The identification numbers and structures are described in the fieldbus related documentation and the fieldbus specific C header file.

Structure Information: *RCX_PACKET_GET_SLAVE_CONN_INFO_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	8+sizeof(*slave data*) 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F0B	RCX_GET_SLAVE_CONN_INFO_CNF
Data			
ulHandle	UINT32	0 … 0xFFFFFFFF	Slave Handle
ulStructID	UINT32	0 … 0xFFFFFFFF	Structure Identification Number
slave data	Structure	n	Fieldbus Specific Slave Status Information (Refer to Fieldbus Documentation)

Packet structure reference

```
/* GET SLAVE CONNECTION INFORMATION CONFIRMATION */
#define RCX_GET_SLAVE_CONN_INFO_CNF          RCX_GET_SLAVE_CONN_INFO_REQ+1

typedef struct RCX_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_Ttag
{
  UINT32  ulHandle;                /* slave handle                      */
  UINT32  ulStructID;              /* structure identification number   */
  /* fieldbus specific slave status information follows here            */
} RCX_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_T;

typedef struct RCX_PACKET_GET_SLAVE_CONN_INFO_CNF_Ttag
{
  RCX_PACKET_HEADER                              tHead;    /* packet header     */
  RCX_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_T    tData;    /* packet data  */
} RCX_PACKET_GET_SLAVE_CONN_INFO_CNF_T;
```

Fieldbus Specific Slave Status Information

The structure returned in the confirmation contains at least a field that helps to unambiguously identify the node. Usually it’s a network address, like MAC ID, IP address or station address. If applicable, the structure may hold a name string.

For details consult the corresponding protocol stack interface manual.

5.5 Protocol Stack Notifications / Indications

Protocol stacks are able to create notifications / indications in form of unsolicited data telegrams exchanged via the mailbox system. These telegrams are used to inform an application about state changes and other protocol stack relevant information.

This section describes the method on how to register / unregister an application with a protocol stack in order to activate and receive unsolicited data telegrams (notifications / indications), via the mailbox system.

Note: It is protocol stack depending which information are available as Notifications / Indications. Please consult the corresponding protocol stack interface manual.

Notifications are automatically activated during the application registration and from this point the application has to process incomming notification packets. If an application does not process the notification / indication telegrams after registration, the protocol stack internal service will time-out which could result into network failures.

If an application registers, *ulSrc* (the *Source Queue Handle*) of the register command is used to identify the host application. It is also stored to verify if further registration / unregistration attemps are valid and *ulSrc* is copied into every notification / indication packet send to the host application to help identifying the intended receiver.

Note: Only one application is able to register with the protocol stack at a time. Furter register attempts in parallel are rejected.

5.5.1 Register Application

Register Application Request

The application uses the following packet in order to register itself with a protocol stack. The packet is send through the channel mailbox.

Structure Information: *RCX_REGISTER_APP_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulSrc	UINT32	n	Unique application identifier
ulCmd	UINT32	0x00002F10	RCX_REGISTER_APP_REQ

Packet structure reference

```
/* REGISTER APPLICATION REQUEST */
#define RCX_REGISTER_APP_REQ                0x00002F10

typedef struct RCX_REGISTER_APP_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
} RCX_REGISTER_APP_REQ_T;
```

Register Application Confirmation

The system channel returns the following packet.

Structure Information: *RCX_REGISTER_APP_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F11	RCX_REGISTER_APP_CNF

Packet structure reference

```
/* REGISTER APPLICATION CONFIRMATION */
#define RCX_REGISTER_APP_CNF                RCX_REGISTER_APP_REQ+1

typedef struct RCX_REGISTER_APP_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
} RCX_REGISTER_APP_CNF_T;
```

5.5.2 Unregister Application

Unregister Application Request

The application uses the following packet in order to undo the registration from above. The packet is send through the channel mailbox.

Structure Information: *RCX_UNREGISTER_APP_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulSrc	UINT32	n	used during registration
ulCmd	UINT32	0x00002F12	RCX_UNREGISTER_APP_REQ

Packet structure reference

```
/* UNREGISTER APPLICATION REQUEST */
#define RCX_UNREGISTER_APP_REQ               0x00002F12

typedef struct RCX_UNREGISTER_APP_REQ_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
} RCX_UNREGISTER_APP_REQ_T;
```

Unregister Application Confirmation

The system channel returns the following packet.

Structure Information: *RCX_UNREGISTER_APP_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See Below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F13	RCX_UNREGISTER_APP_CNF

Packet structure reference

```
/* UNREGISTER APPLICATION CONFIRMATION */
#define RCX_UNREGISTER_APP_CNF               RCX_UNREGISTER_APP_REQ+1

typedef struct RCX_UNREGISTER_APP_CNF_Ttag
{
  RCX_PACKET_HEADER                tHead;   /* packet header             */
} RCX_UNREGISTER_APP_CNF_T;
```

5.6 Link Status Changed Service

This service is used to inform an application about about link status changes of a protocol stack. In order to receive the notifications, the application has to register itself at the protocol stack (see *5.5 Protocol Stack Notifications / Indications*).

Note: This command depends on the used protocol stack. Consult the corresponding protocol stack interface manual if the command is supported and for more information.

Link Status Change Indication

Structure Information: *RCX_LINK_STATUS_CHANGE_IND_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	32	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F8A	RCX_LINK_STATUS_CHANGE_IND
Data			
atLinkData[2]	Structure		Link Status Information

Structure Information: *RCX_LINK_STATUS_T*			
ulPort	UINT32		Number of the port
fIsFullDuplex	UINT32		Non-zero if full duplex is used
fIsLinkUp	UINT32		Non-zero if link is up
ulSpeed	UINT32	0 10 100	Speed of the link No link 10MBit 100Mbit

Packet structure reference

```
/* LINK STATUS CHANGE INDICATION */
#define RCX_LINK_STATUS_CHANGE_IND         0x00002F8A

typedef struct RCX_LINK_STATUS_Ttag
{
  UINT32        ulPort;           /*!< Port the link status is for */
  UINT32        fIsFullDuplex;    /*!< If a full duplex link is available on this port */
  UINT32        fIsLinkUp;        /*!< If a link is available on this port */
  UINT32        ulSpeed;          /*!< Speed of the link \n\n
                                     \valueRange
                                     0:   No link \n
                                     10:  10MBit  \n
                                     100: 100MBit \n */
} RCX_LINK_STATUS_T;

typedef struct RCX_LINK_STATUS_CHANGE_IND_DATA_Ttag
{
  RCX_LINK_STATUS_T  atLinkData[2];
} RCX_LINK_STATUS_CHANGE_IND_DATA_T;

typedef struct RCX_LINK_STATUS_CHANGE_IND_Ttag
{
  RCX_PACKET_HEADER                 tHead;
  RCX_LINK_STATUS_CHANGE_IND_DATA_T tData;
} RCX_LINK_STATUS_CHANGE_IND_T;
```

Link Status Change Response

Structure Information: *RCX_LINK_STATUS_CHANGE_RES_T*			
Variable	**Type**	**Value / Range**	**Description**
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F8B	RCX_LINK_STATUS_CHANGE_RES

Packet structure reference

```
/* LINK STATUS CHANGE RESPONSE */
#define RCX_LINK_STATUS_CHANGE_RES         RCX_LINK_STATUS_CHANGE_IND+1

typedef RCX_PACKET_HEADER      RCX_LINK_STATUS_CHANGE_RES_T;
```

5.7 Perform a Bus Scan

Perform a bus scan and retrieve the scan results. This services in only offered by master protocol stacks.

Note: This command depends on the used protocol stack. Consult the corresponding protocol stack interface manual if the command is supported and for more information.

Bus Scan Request

Structure Information: *RCX_BUSSCAN_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F22	RCX_BUSSCAN_REQ
Data			
ulAction	UINT32	0x01 0x02 0x03	Action to perform RCX_BUSSCAN_CMD_START RCX_BUSSCAN_CMD_STATUS RCX_BUSSCAN_CMD_ABORT

Packet structure reference

```
/* BUS SCAN REQUEST */
#define RCX_BUSSCAN_REQ                    0x00002F22

#define RCX_BUSSCAN_CMD_START    0x01
#define RCX_BUSSCAN_CMD_STATUS   0x02
#define RCX_BUSSCAN_CMD_ABORT    0x03

typedef struct RCX_BUSSCAN_REQ_DATA_Ttag
{
  UINT32 ulAction;
} RCX_BUSSCAN_REQ_DATA_T;

typedef struct RCX_BUSSCAN_REQ_Ttag
{
  RCX_PACKET_HEADER       tHead;
  RCX_BUSSCAN_REQ_DATA_T  tData;
} RCX_BUSSCAN_REQ_T;
```

Bus Scan Confirmation

Structure Information: *RCX_BUSSCAN_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	12 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F23	RCX_BUSSCAN_CNF
Data			
ulMaxProgress	UINT32	n	Number of devices from the configuration
ulActProgress	UINT32	m	Number of devices found
abDevice List[4]	UINT8		List of available devices on the fieldbus system

Packet structure reference

```
/* BUS SCAN CONFIRMATION */
#define RCX_BUSSCAN_CNF                     RCX_BUSSCAN_REQ+1

typedef struct RCX_BUSSCAN_CNF_DATA_Ttag
{
  UINT32 ulMaxProgress;
  UINT32 ulActProgress;
  UINT8  abDeviceList[4];
} RCX_BUSSCAN_CNF_DATA_T;

typedef struct RCX_BUSSCAN_CNF_Ttag
{
  RCX_PACKET_HEADER      tHead;
  RCX_BUSSCAN_CNF_DATA_T tData;
} RCX_BUSSCAN_CNF_T;
```

5.8 Get Information about a Fieldbus Device

Read the available information about a specific node on the fieldbus system. This services in only offered by master protocol stacks.

Note: This command depends on the used protocol stack. Consult the corresponding protocol stack interface manual if the command is supported and for more information.

Get Device Info Request

Structure Information: *RCX_GET_DEVICE_INFO_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulLen	UINT32	4	Packet Data Length (in Bytes)
ulCmd	UINT32	0x00002F24	RCX_GET_DEVICE_INFO_REQ
Data			
ulDeviceIdx	UINT32	n	Fieldbus specific device identifier

Packet structure reference

```
/* GET DEVICE INFO REQUEST */
#define RCX_GET_DEVICE_INFO_REQ            0x00002F24

typedef struct RCX_GET_DEVICE_INFO_REQ_DATA_Ttag
{
  UINT32 ulDeviceIdx;
} RCX_GET_DEVICE_INFO_REQ_DATA_T;

typedef struct RCX_GET_DEVICE_INFO_REQ_Ttag
{
  RCX_PACKET_HEADER              tHead;
  RCX_GET_DEVICE_INFO_REQ_DATA_T tData;
} RCX_GET_DEVICE_INFO_REQ_T;
```

Get Device Info Confirmation

Structure Information: *RCX_GET_DEVICE_INFO_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	8 + n 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F25	RCX_GET_DEVICE_INFO_CNF
Data			
ulDeviceIdx	UINT32	n	Identifier of device
ulStructId	UINT32	m	Identifier of structure type
	Structure		Fieldbus specific data structure

Packet structure reference

```
/* GET DEVICE INFO CONFIRMATION */
#define RCX_GET_DEVICE_INFO_CNF            RCX_GET_DEVICE_INFO_REQ+1

typedef struct RCX_GET_DEVICE_INFO_CNF_DATA_Ttag
{
  UINT32 ulDeviceIdx;
  UINT32 ulStructId;
  /* UINT8  tStruct; Fieldbus specific structure */
} RCX_GET_DEVICE_INFO_CNF_DATA_T;

typedef struct RCX_GET_DEVICE_INFO_CNF_Ttag
{
  RCX_PACKET_HEADER              tHead;
  RCX_GET_DEVICE_INFO_CNF_DATA_T tData;
} RCX_GET_DEVICE_INFO_CNF_T;
```

5.9 Configuration in Run

Configuration in Run is a fieldbus and protocol stack specific function which should allow the modification of the master fieldbus configuration while the configuration is active and without stopping the already active bus communication. The functions only works if a configuration database file is used to configure the master device.

The modification of configuration data during run-time has some specific limitations. Therefore the modified configuration database must first be downloaded to the master device. Afterwards the master is requested to check if the new configuration database can be used without disturbing the current active devices on the fieldbus system (e.g. adding a new device online).

Note: This command depends on the used protocol stack and not all fieldbus systems are supporting *Configuration in Run*.
Consult the corresponding protocol stack interface manual if this function is supported and about additional information on how to use the function.

5.9.1 Verify Configuration Database

This packet informs the master, that a new configuration database file was downloaded and available to be verified.

Verify Database Request

Structure Information: *RCX_VERIFY_DATABASE_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	UINT32	0x00002F82	RCX_VERIFY_DATABASE_REQ

Packet structure reference

```
/* VERIFY DATABASE REQUEST */
#define RCX_VERIFY_DATABASE_REQ            0x00002F82

typedef struct RCX_VERIFY_DATABASE_REQ_Ttag
{
  RCX_PACKET_HEADER tHead;                    /* packet header */
} RCX_VERIFY_DATABASE_REQ_T;
```

Verify Database Confirmation

Structure Information: *RCX_VERIFY_DATABASE_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	116 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F83	RCX_VERIFY_DATABASE_CNF
Data			
tNewSlaves	Structure	n	Addresses of new slaves which have to be configured.
tDeactivated Slaves	Structure	n	Addresses of slaves which are deactivated or cannot be configured.
tChanged Slaves	Structure	n	Addresses of slaves whose configuration has been changed.
tUnchanged Slaves	Structure	n	Addresses of slaves whose configuration has not been changed.
tImpossible SlaveChanges	Structure	n	Addresses of slaves whose configuration is not valid.
tMaster Changes	Structure	n	Field bus changes and status.

Packet structure reference

```
/* VERIFY DATABASE CONFIRMATION */
#define RCX_VERIFY_DATABASE_CNF            RCX_VERIFY_DATABASE_REQ+1

typedef struct RCX_VERIFY_SLAVE_DATABASE_LIST_Ttag
{
    UINT32  ulLen;
    UINT8   abData[16];
} RCX_VERIFY_SLAVE_DATABASE_LIST_T;

typedef struct RCX_VERIFY_MASTER_DATABASE_Ttag
{
  UINT32  ulMasterSettings;   /* field bus independent changes */
  UINT32  ulMasterStatus;     /* field bus specific status */
  UINT32  ulReserved[2];
} RCX_VERIFY_MASTER_DATABASE_T;

#define RCX_CIR_MST_SET_STARTUP       0x00000001
#define RCX_CIR_MST_SET_WATCHDOG      0x00000002
#define RCX_CIR_MST_SET_STATUSOFFSET  0x00000004
#define RCX_CIR_MST_SET_BUSPARAMETER  0x00000008

typedef struct RCX_VERIFY_DATABASE_CNF_DATA_Ttag
{
    RCX_VERIFY_SLAVE_DATABASE_LIST_T               tNewSlaves;
    RCX_VERIFY_SLAVE_DATABASE_LIST_T               tDeactivatedSlaves;
    RCX_VERIFY_SLAVE_DATABASE_LIST_T               tChangedSlaves;
    RCX_VERIFY_SLAVE_DATABASE_LIST_T               tUnchangedSlaves;
    RCX_VERIFY_SLAVE_DATABASE_LIST_T               tImpossibleSlaveChanges;
    RCX_VERIFY_MASTER_DATABASE_T                   tMasterChanges;
} RCX_VERIFY_DATABASE_CNF_DATA_T;

typedef struct RCX_VERIFY_DATABASE_CNF_Ttag
{
  RCX_PACKET_HEADER                        tHead;    /* packet header  */
  RCX_VERIFY_DATABASE_CNF_DATA_T           tData;    /* packet data    */
} RCX_VERIFY_DATABASE_CNF_T;
```

5.9.2 Activate Configuration Database

This packet indicates the master to activate the new configuration.

Activate Database Request

Structure Information: *RCX_ACTIVATE_DATABASE_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	UINT32	0x00000020	RCX_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	UINT32	0x00002F84	RCX_ACTIVATE_DATABASE_REQ

Packet structure reference

```
/* ACTIVATE DATABASE REQUEST */
#define RCX_ACTIVATE_DATABASE_REQ          0x00002F84

typedef struct RCX_ACTIVATE_DATABASE_REQ_Ttag
{
  RCX_PACKET_HEADER tHead;                    /* packet header */
} RCX_ACTIVATE_DATABASE_REQ_T;
```

Activate Database Confirmation

Structure Information: *RCX_ACTIVATE_DATABASE_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulLen	UINT32	16 0	Packet Data Length (in Bytes) If ulState = RCX_S_OK Otherwise
ulState	UINT32	See below	Status / Error Code, see Section 6
ulCmd	UINT32	0x00002F85	RCX_ACTIVATE_DATABASE_CNF
Data			
abSlvSt[16]	UINT8	n	State of the Slaves after Configuration

Packet structure reference

```
/* ACTIVATE DATABASE CONFIRMATION */
#define RCX_ACTIVATE_DATABASE_CNF          RCX_ACTIVATE_DATABASE_REQ+1

typedef struct RCX_ACTIVATE_DATABASE_CNF_DATA_Ttag
{
  UINT8                abSlvSt[16];  /* State of the slaves after configuration */
} RCX_ACTIVATE_DATABASE_CNF_DATA_T;

typedef struct RCX_ACTIVATE_DATABASE_CNF_Ttag
{
  RCX_PACKET_HEADER                        tHead;  /* packet header */
  RCX_ACTIVATE_DATABASE_CNF_DATA_T         tData;
} RCX_ACTIVATE_DATABASE_CNF_T;
```

6 Status and error codes

The following status and error codes may be returned in *ulState* of the packet. Not all of the codes outlined below are supported by a specific protocol stack.

6.1 Packet error codes

Value	Definition / Description
0x00000000	RCX_S_OK Success, Status Okay
0xC0000001	RCX_E_FAIL Fail
0xC0000002	RCX_E_UNEXPECTED Unexpected
0xC0000003	RCX_E_OUTOFMEMORY Out Of Memory
0xC0000004	RCX_E_UNKNOWN_COMMAND Unknown Command
0xC0000005	RCX_E_UNKNOWN_DESTINATION Unknown Destination
0xC0000006	RCX_E_UNKNOWN_DESTINATION_ID Unknown Destination ID
0xC0000007	RCX_E_INVALID_PACKET_LEN Invalid Packet Length
0xC0000008	RCX_E_INVALID_EXTENSION Invalid Extension
0xC0000009	RCX_E_INVALID_PARAMETER Invalid Parameter
0xC000000C	RCX_E_WATCHDOG_TIMEOUT Watchdog Timeout
0xC000000D	RCX_E_INVALID_LIST_TYPE Invalid List Type
0xC000000E	RCX_E_UNKNOWN_HANDLE Unknown Handle
0xC000000F	RCX_E_PACKET_OUT_OF_SEQ Out Of Sequence
0xC0000010	RCX_E_PACKET_OUT_OF_MEMORY Out Of Memory
0xC0000011	RCX_E_QUE_PACKETDONE Queue Packet Done
0xC0000012	RCX_E_QUE_SENDPACKET Queue Send Packet
0xC0000013	RCX_E_POOL_PACKET_GET Pool Packet Get
0xC0000015	RCX_E_POOL_GET_LOAD Pool Get Load
0xC000001A	RCX_E_REQUEST_RUNNING Request Already Running
0xC0000100	RCX_E_INIT_FAULT Initialization Fault
0xC0000101	RCX_E_DATABASE_ACCESS_FAILED Database Access Failed
0xC0000119	RCX_E_NOT_CONFIGURED Not Configured

Value	Definition / Description
0xC0000120	RCX_E_CONFIGURATION_FAULT Configuration Fault
0xC0000121	RCX_E_INCONSISTENT_DATA_SET Inconsistent Data Set
0xC0000122	RCX_E_DATA_SET_MISMATCH Data Set Mismatch
0xC0000123	RCX_E_INSUFFICIENT_LICENSE Insufficient License
0xC0000124	RCX_E_PARAMETER_ERROR Parameter Error
0xC0000125	RCX_E_INVALID_NETWORK_ADDRESS Invalid Network Address
0xC0000126	RCX_E_NO_SECURITY_MEMORY No Security Memory
0xC0000140	RCX_E_NETWORK_FAULT Network Fault
0xC0000141	RCX_E_CONNECTION_CLOSED Connection Closed
0xC0000142	RCX_E_CONNECTION_TIMEOUT Connection Timeout
0xC0000143	RCX_E_LONELY_NETWORK Lonely Network
0xC0000144	RCX_E_DUPLICATE_NODE Duplicate Node
0xC0000145	RCX_E_CABLE_DISCONNECT Cable Disconnected
0xC0000180	RCX_E_BUS_OFF Network Node Bus Off
0xC0000181	RCX_E_CONFIG_LOCKED Configuration Locked
0xC0000182	RCX_E_APPLICATION_NOT_READY Application Not Ready
0xC002000C	RCX_E_TIMER_APPL_PACKET_SENT Timer App Packet Sent
0xC02B0001	RCX_E_QUE_UNKNOWN Unknown Queue
0xC02B0002	RCX_E_QUE_INDEX_UNKNOWN Unknown Queue Index
0xC02B0003	RCX_E_TASK_UNKNOWN Unknown Task
0xC02B0004	RCX_E_TASK_INDEX_UNKNOWN Unknown Task Index
0xC02B0005	RCX_E_TASK_HANDLE_INVALID Invalid Task Handle
0xC02B0006	RCX_E_TASK_INFO_IDX_UNKNOWN Unknown Index
0xC02B0007	RCX_E_FILE_XFR_TYPE_INVALID Invalid Transfer Type
0xC02B0008	RCX_E_FILE_REQUEST_INCORRECT Invalid File Request
0xC02B000E	RCX_E_TASK_INVALID Invalid Task
0xC02B001D	RCX_E_SEC_FAILED Security EEPROM Access Failed

Value	Definition / Description
0xC02B001E	RCX_E_EEPROM_DISABLED EEPROM Disabled
0xC02B001F	RCX_E_INVALID_EXT Invalid Extension
0xC02B0020	RCX_E_SIZE_OUT_OF_RANGE Block Size Out Of Range
0xC02B0021	RCX_E_INVALID_CHANNEL Invalid Channel
0xC02B0022	RCX_E_INVALID_FILE_LEN Invalid File Length
0xC02B0023	RCX_E_INVALID_CHAR_FOUND Invalid Character Found
0xC02B0024	RCX_E_PACKET_OUT_OF_SEQ Packet Out Of Sequence
0xC02B0025	RCX_E_SEC_NOT_ALLOWED Not Allowed In Current State
0xC02B0026	RCX_E_SEC_INVALID_ZONE Security EEPROM Invalid Zone
0xC02B0028	RCX_E_SEC_EEPROM_NOT_AVAIL Security EEPROM Not Available
0xC02B0029	RCX_E_SEC_INVALID_CHECKSUM Security EEPROM Invalid Checksum
0xC02B002A	RCX_E_SEC_ZONE_NOT_WRITEABLE Security EEPROM Zone Not Writeable
0xC02B002B	RCX_E_SEC_READ_FAILED Security EEPROM Read Failed
0xC02B002C	RCX_E_SEC_WRITE_FAILED Security EEPROM Write Failed
0xC02B002D	RCX_E_SEC_ACCESS_DENIED Security EEPROM Access Denied
0xC02B002E	RCX_E_SEC_EEPROM_EMULATED Security EEPROM Emulated
0xC02B0038	RCX_E_INVALID_BLOCK Invalid Block
0xC02B0039	RCX_E_INVALID_STRUCT_NUMBER Invalid Structure Number
0xC02B4352	RCX_E_INVALID_CHECKSUM Invalid Checksum
0xC02B4B54	RCX_E_CONFIG_LOCKED Configuration Locked
0xC02B4D52	RCX_E_SEC_ZONE_NOT_READABLE Security EEPROM Zone Not Readable

Table 22: Status and error codes

7 Appendix

7.1 List of figures

Figure 1: Flowchart File Download 38
Figure 2: Flowchart File Upload 44
Figure 3: Flowchart Modify Configuration Settings 100

7.2 List of tables

Table 1: List of revisions 4
Table 2: Terms, abbreviations and definitions 5
Table 3: References to documents 5
Table 4: General packet structure 7
Table 5: Boot Type 17
Table 6: Chip Type 17
Table 7: Set MAC Address Parameter Field 33
Table 8: Set MAC Address Parameter Field 33
Table 9: Sub Block Type 60
Table 10: Transmission Flags 60
Table 11: Hand Shake Mode 61
Table 12: Hardware Configuration (Zone 1) 63
Table 13: PCI System and OS Setting (Zone 2) 63
Table 14: User Specific Zone (Zone 3) 63
Table 15: Time Command Field 73
Table 16: Clock Status 74
Table 17: Clock Status 74
Table 18: RCX_SET_HANDSHAKE_CONFIG_REQ_T - Set Handshake Configuration 97
Table 19: RCX_SET_FW_PARAMETER_REQ_T – Set Parameter Data 102
Table 20: Encoding Parameter Identifier 102
Table 21: Defined Parameter Identifier 103
Table 22: Status and error codes 124

7.3 Legal notes

Copyright

Important notes

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

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

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

Liability disclaimer

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

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

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

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

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

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

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

Warranty

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

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

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

Costs of support, maintenance, customization and product care

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

Additional guarantees

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

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

Confidentiality

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

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

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

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

Export provisions

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

7.4 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