Packet API

netX Dual-Port Memory

Packet-based services (netX 10/50/51/52/100/500)

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC161001API05EN | Revision 5 | English | 2021-09 | Released | Public

Table of content

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
- focus on the available system services, their functionality and definitions.

1.2 List of revisions

Rev	Date	Name	Revisions
2	2018-11-13	BME, HHE	Section *File Download Data*: ulLen updated.
			Section *General packet fragmentation* added.
			Sections *Read Common Control Block*, *Read Common Status Block*, and *Read Extended Status Block*: Details about ulChannelId added.
			Section *Read I/O Process Data Image Size*: Information about additional length added.
			Section *Channel Initialization*: Figure 3 added.
			Section *Protocol Stack Notifications / Indications* and *Link Status Changed Service*: Information added.
3	2019-07-21	AM	Section *Set MAC Address*: Storing of MAC in FDL is not supported.
			Section *Delete Protocol Stack Configuration*: Handling for remanent data added.
4	2020-06-02	AM, RMA, HHE	Exchanged unknown definitions HIL_COMM_CHANNEL_xx to HIL_FILE_CHANNEL_xx.
			Section *Files and folders*: ulId and fragmentation Handling
			Section *Identifying netX Hardware:* Default value for serial number is 8194.
			Fixed command definition in HIL_START_STOP_COMM_REQ (HIL_START_STOP_COMM_PARAM_START / _STOP)
			Section *Read Extended Status Block*: Fragmentation is not supported for this service.
5	2021-09-13	ALM, RMA	Section *Downloading / Uploading Files*: Multi packet example ulId handling updated.
			Section *File Download Data*: Confirmation always returns 4 byte (ulLen = 4)
			Section *Read Hardware Identification Data*: Boot Type and Chip Type updated.
			Section *Device Data Provider (DDP)* added.

Table 1: List of revisions

1.3 Terms, abbreviations and definitions

Term	Description
DPM	Dual-port memory
FW	Firmware
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 17, English.

[2] Hilscher Gesellschaft für Systemautomation mbH: Function Description, Second Stage Boot Loader, netX 10/50/51/52/100/500, V1.6, Revision 17, 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.

HIL_Packet.h	Provides the “Packet” structure
HIL_SystemCmd.h	Provides the system commands and structures
HIL_ApplicationCmd.h	Provides the commands and structures for the application task
HIL_DualPortMemory.h	Provides the netX dual-port memory layout

For using protocol-specific functions, you need further header files provided by the protocol stack: Protocol-specific header files are coming with the firmware implementation and using additional header files.

Due to further development and standardization, new header files were introduced:
file names HIL_*.h.

Prefix RCX_ in definitions is replaced by the prefix HIL_.

Header file `rcx_Public.h` is replaced by `HIL_Packet.h`, `HIL_SystemCmd.h`, and `HIL_ApplicationCmd.h`.

Header file `rcx_User.h` is replaced by `HIL_DualPortMemory.h`.

2.1 General packet structure

The structure HIL_PACKET_T is the general structure of a packet. The description is a short extract from the information in the *netX Dual-Port Memory Interface Manual* (reference [1]).

Area	Variable / Element	Type	Value / Range	Description
Header (tHead)	ulDest	uint32_t	0 ... 0xFFFFFFFF	Destination Address / Handle
	ulSrc	uint32_t	0 ... 0xFFFFFFFF	Source Address / Handle
	ulDestId	uint32_t	0 ... 0xFFFFFFFF	Destination Identifier
	ulSrcId	uint32_t	0 ... 0xFFFFFFFF	Source Identifier
	ulLen	uint32_t	0 ... max. packet data size	Packet Data Length (in byte)
	ulId	uint32_t	0 ... 0xFFFFFFFF	Packet Identifier
	ulSta	uint32_t	0 ... 0xFFFFFFFF	Packet State / Error
	ulCmd	uint32_t	0 ... 0xFFFFFFFF	Packet Command / Confirmation
	ulExt	uint32_t	0 or extension bit mask	Packet Extension
	ulRout	uint32_t	0 ... 0xFFFFFFFF	Reserved (routing information)
Packet Data (abData)	abData	…	0 … 0xFF	Packet Data (packet-specific data)

Table 4: General packet structure: HIL_PACKET_T

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.

Variable / Element	Brief description
ulDest ulDestId ulSrc ulSrcId	**Destination Address / Handle** **Destination Identifier** **Source Address / Handle** **Source Identifier** These elements are used to address the receiver and sender of a packet.
ulLen	**Packet Data Length** ulLen defines how many data follow the packet header. The length is counted **in bytes**. The packet header length is not included in ulLen and has a fixed length of 40 bytes (see HIL_PACKET_HEADER_T)
ulId	**Packet Identifier** 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.
ulSta	**Packet State / Error** ulSta is used to signal packet errors in an answer (response/confirmation) packet. The value is 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. ERR_HIL_UNKNOWN_COMMAND, ERR_HIL_INVALID_PACKET_LEN, ERR_HIL_PARAMETER_ERROR etc.*)*

ulCmd	**Packet Command / Packet Answer** 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 netX-based device ▪ HIL_HW_IDENTIFY_REQ (0x00001EB8) Command to read general hardware information like device number / serial number etc. ▪ HIL_HW_IDENTIFY_CNF (0x00001EB9) Answer to the HIL_HW_IDENTIFY_REQ command.
ulExt	**Packet Extension** ulExt is used to mark packets as packets of a sequence, in case a transfer consists of multiple packets (e.g. file download).
ulRout	**Reserved (Routing Information)** This is reserved for further use (shall not be changed by the receiver of a packet).
abData	**Packet Data** 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.

Table 5: Brief description of the elements/variables of a 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 & HIL_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 `ulSta`of the answer packet to be 0 before evaluating the packet data, `ulSta` 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 __HIL_PACKED_PRE struct HIL_FILE_GET_MD5_REQ_DATA_Ttag
{
  uint32_t                              ulChannelNo;            /* 0 = Channel 0, ..., 3
= Channel 3, 0xFFFFFFFF = System, see HIL_FILE_xxxx */
  uint16_t                              usFileNameLength;       /* length of NUL-
terminated file name that will follow */
  /* a NUL-terminated file name will follow here */
} __HIL_PACKED_POST HIL_FILE_GET_MD5_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct HIL_FILE_GET_MD5_REQ_Ttag
{
  HIL_PACKET_HEADER_T                     tHead;                  /* packet header */
  HIL_FILE_GET_MD5_REQ_DATA_T             tData;                  /* packet data */
} __HIL_PACKED_POST HIL_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`.

```
HIL_PACKET                    tPacket;
HIL_FILE_GET_MD5_REQ_DATA_T* ptMD5Data  = (HIL_FILE_GET_MD5_REQ_DATA_T*)tPacket.abData;
uint8_t*                       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_t*)(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 netX operating system 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	HIL_FIRMWARE_RESET_REQ	13
Identification and information		
Read the general hardware identification information	HIL_HW_IDENTIFY_REQ	14
Read the device-specific hardware information	HIL_HW_HARDWARE_INFO_REQ	18
Read the name and version or firmware, operating system or protocol stack running on a communication channel	HIL_FIRMWARE_IDENTIFY_REQ	20
System Channel Information Blocks		
Read the system channel *System Information Block*	HIL_SYSTEM_INFORMATION_BLOCK_REQ	25
Read the system channel *Channel Information Block*	HIL_CHANNEL_INFORMATION_BLOCK_REQ	26
Read the system channel *System Control Block*	HIL_SYSTEM_CONTROL_BLOCK_REQ	29
Read the system channel *System Status Block*	HIL_SYSTEM_STATUS_BLOCK_REQ	30
MAC Address Handling		
Set / store a new MAC address on the device	HIL_SET_MAC_ADDR_REQ	31
Files and folders		
List directories and files from the file system	HIL_DIR_LIST_REQ	36
Download a file (start, send file data, abort)	HIL_FILE_DOWNLOAD_REQ	40
	HIL_FILE_DOWNLOAD_DATA_REQ	43
	HIL_FILE_DOWNLOAD_ABORT_REQ	45
File Upload (start, read file data, abort)	HIL_FILE_UPLOAD_REQ	47
	HIL_FILE_UPLOAD_DATA_REQ	49
	HIL_FILE_UPLOAD_ABORT_REQ	51
File Delete	HIL_FILE_DELETE_REQ	52
File Rename	HIL_FILE_RENAME_REQ	54
Create a CRC32 checksum	(example code)	56
Calculate the MD5 checksum for a given file	HIL_FILE_GET_MD5_REQ	57
Read the MD5 checksum from the file header of a given file	HIL_FILE_GET_HEADER_MD5_REQ	59

License Information		
Read the license information stored on the netX hardware	HIL_HW_LICENSE_INFO_REQ	80
Determining the DPM Layout		
Read and evaluate the DPM Layout of the system / communication channels	HIL_DPM_GET_BLOCK_INFO_REQ	60
Security Memory / Flash Device Label		
Read device-specific data from Security Memory / Flash Device Label	HIL_SECURITY_EEPROM_READ_REQ	76
Write device-specific data to Security Memory / Flash Device Label	HIL_SECURITY_EEPROM_WRITE_REQ	78
System Performance Counter		
Read the firmware performance counters	HIL_GET_PERF_COUNTERS_REQ	81
Real-Time Clock		
Read / set the real-time clock if available	HIL_TIME_COMMAND_REQ	83
Start RAM based Firmware		
Start a RAM-based firmware which was downloaded before	HIL_CHANNEL_INSTANTIATE_REQ	86
Second Stage Bootloader (only)		
Format the default partition containing the file system	HIL_FORMAT_REQ	88

Table 6: System services (function overview)

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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	8	Packet data length (in Bytes)
ulCmd	uint32_t	0x00001E00	HIL_FIRMWARE_RESET_REQ
Data			
ulTimeToReset	uint32_t	0	Time delay until reset is executed in milliseconds [ms] Fix: 500ms (not changeable)
ulResetMode	uint32_t	0	Reset Mode (not used, set to zero)

Table 7: HIL_FIRMWARE_RESET_REQ_T – Firmware Reset request

Packet structure reference

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

typedef struct HIL_FIRMWARE_RESET_REQ_DATA_Ttag
{
  uint32_t ulTimeToReset;  /* time to reset in ms   */
  uint32_t ulResetMode;    /* reset mode parameter  */
} HIL_FIRMWARE_RESET_REQ_DATA_T;

typedef struct HIL_FIRMWARE_RESET_REQtag
{
  HIL_PACKET_HEADER              tHead;  /* packet header */
  HIL_FIRMWARE_RESET_REQ_DATA_T  tData;  /* packet data   */
} HIL_FIRMWARE_RESET_REQ_T;
```

Firmware Reset confirmation

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / error code, see Section 6.
ulCmd	uint32_t	0x00001E01	HIL_CHANNEL_RESET_CNF

Table 8: HIL_FIRMWARE_RESET_CNF_T – Firmware Reset confirmation

Packet structure reference

```
/* CHANNEL RESET CONFIRMATION */
#define HIL_CHANNEL_RESET_CNF               HIL_CHANNEL_RESET_REQ+1

typedef struct HIL_FIRMWARE_RESET_CNF_Ttag
{
  HIL_PACKET_HEADER    tHead;  /* packet header                          */
} HIL_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 8194
- 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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001EB8	HIL_HW_IDENTIFY_REQ

Table 9: HIL_HW_IDENTIFY_REQ_T – Hardware Identify request

Packet structure reference

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

typedef struct HIL_HW_IDENTIFY_REQ_Ttag
{
  HIL_PACKET_HEADER     tHead;             /* packet header                */
} HIL_HW_IDENTIFY_REQ_T;
```

Hardware Identify confirmation

The channel firmware returns the following packet.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 36 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EB9	HIL_HW_IDENTIFY_CNF
Data			
ulDeviceNumber	uint32_t	0 … 0xFFFFFFFF	Device Number
ulSerialNumber	uint32_t	0 … 0xFFFFFFFF	Serial Number
ausHwOptions[4]	uint16_t	0 … 0xFFFF	Hardware Assembly Option
usDeviceClass	uint16_t	0 … 0xFFFF	netX Device Class
bHwRevision	uint8_t	0 … 0xFF	Hardware Revision Index
bHwCompatibility	uint8_t	0 … 0xFF	Hardware Compatibility Index
ulBootType	uint32_t	0 … 8	Hardware Boot Type
ulChipType	uint32_t	0 … n	Chip Type (see tables below)
ulChipStep	uint32_t	0 … 0x000000FF	Chip Step
ulRomcodeRevision	uint32_t	0 … 0x00000FFF	ROM Code Revision

Table 10: HIL_HW_IDENTIFY_CNF_T – Hardware Identify confirmation

Packet structure reference

```
/* HARDWARE IDENTIFY CONFIRMATION */
#define HIL_HW_IDENTIFY_CNF                  HIL_HW_IDENTIFY_REQ+1

typedef struct HIL_HW_IDENTIFY_CNF_DATA_Ttag
{
  uint32_t ulDeviceNumber;              /* device number / identification */
  uint32_t ulSerialNumber;              /* serial number                  */
  uint16_t ausHwOptions[4];             /* hardware options               */
  uint16_t usDeviceClass;               /* device class                   */
  uint8_t  bHwRevision;                 /* hardware revision              */
  uint8_t  bHwCompatibility;            /* hardware compatibility         */
  uint32_t ulBootType;                  /* boot type                      */
  uint32_t ulChipTyp;                   /* chip type                      */
  uint32_t ulChipStep;                  /* chip step                      */
  uint32_t ulRomcodeRevision;           /* rom code revision              */
} HIL_HW_IDENTIFY_CNF_DATA_T;

typedef struct HIL_HW_IDENTIFY_CNF_Ttag
{
  HIL_PACKET_HEADER           tHead;    /* packet header                  */
  HIL_HW_IDENTIFY_CNF_DATA_T  tData;    /* packet data                    */
} HIL_HW_IDENTIFY_CNF_T;
```

Boot Type

Boot Type

This field indicates from which resource the system was started.

Value	Description	Definition
0x00000000	ROM Loader: PARALLEL FLASH (SRAM Bus)	HIL_DEV_BOOT_TYPE_PFLASH_SRAMBUS
0x00000001	ROM Loader: PARALLEL FLASH (Extension Bus)	HIL_DEV_BOOT_TYPE_PFLASH_EXTBUS
0x00000002	ROM Loader: DUAL-PORT MEMORY	HIL_DEV_BOOT_TYPE_DUALPORT
0x00000003	ROM Loader: PCI INTERFACE	HIL_DEV_BOOT_TYPE_PCI
0x00000004	ROM Loader: MULTIMEDIA CARD	HIL_DEV_BOOT_TYPE_MMC
0x00000005	ROM Loader: I2C BUS	HIL_DEV_BOOT_TYPE_I2C
0x00000006	ROM Loader: SERIAL FLASH	HIL_DEV_BOOT_TYPE_SFLASH
0x00000007	Second Stage Boot Loader: SERIAL FLASH	HIL_DEV_BOOT_TYPE_2ND_STAGE_FLASH_BASED
0x00000008	Second Stage Boot Loader: RAM	HIL_DEV_BOOT_TYPE_2ND_STAGE_RAM_BASED
0x00000009	ROM Loader: Internal Flash	HIL_DEV_BOOT_TYPE_IFLASH
Other values are reserved		-

Table 11: Boot Type

Chip Type

This field indicates the type of chip that is used.

Value	Chip Namen	Definition
0x00000000	Unknown	HIL_DEV_CHIP_TYPE_UNKNOWN
0x00000001	netX 500	HIL_DEV_CHIP_TYPE_NETX500
0x00000002	netX 100	HIL_DEV_CHIP_TYPE_NETX100
0x00000003	netX 50	HIL_DEV_CHIP_TYPE_NETX50
0x00000004	netX 10	HIL_DEV_CHIP_TYPE_NETX10
0x00000005	netX 51	HIL_DEV_CHIP_TYPE_NETX51
0x00000006	netX 52	HIL_DEV_CHIP_TYPE_NETX52
0x00000007	netX 4000	HIL_DEV_CHIP_TYPE_NETX4000
0x00000008	netX 4100	HIL_DEV_CHIP_TYPE_NETX4100
0x00000009	netX 90	HIL_DEV_CHIP_TYPE_NETX90
0x0000000A	netIOL	HIL_DEV_CHIP_TYPE_NETIOL
0x0000000B	netX XXL MPW	HIL_DEV_CHIP_TYPE_NETXXXL_MPW
Other values are reserved		

Table 12: Chip Type

3.4 Read Hardware Information

Hardware Info request

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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001EF6	HIL_HW_HARDWARE_INFO_REQ

Table 13: HIL_HW_HARDWARE_INFO_REQ_T – Hardware Info request

Packet structure reference

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

typedef struct HIL_HW_HARDWARE_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER     tHead;             /* packet header               */
} HIL_HW_HARDWARE_INFO_REQ_T;
```

Hardware Info confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	56 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EF7	HIL_HW_HARDWARE_INFO_CNF
Data			
ulDeviceNumber	uint32_t	0 … 0xFFFFFFFF	Device Number / Identification
ulSerialNumber	uint32_t	0 … 0xFFFFFFFF	Serial Number
ausHwOptions[4]	Array of uint16_t	0 … 0xFFFF	Hardware Assembly Option
usManufacturer	uint16_t	0 … 0xFFFF	Manufacturer Code
usProductionDate	uint16_t	0 … 0xFFFF	Production Date
ulLicenseFlags1	uint32_t	0 … 0xFFFFFFFF	License Flags 1
ulLicenseFlags2	uint32_t	0 … 0xFFFFFFFF	License Flags 2
usNetxLicenseID	uint16_t	0 … 0xFFFF	netX License Identification
usNetxLicenseFlags	uint16_t	0 … 0xFFFF	netX License Flags
usDeviceClass	uint16_t	0 … 0xFFFF	netX Device Class
bHwRevision	uint8_t	0 …0xFFFF	Hardware Revision Index
bHwCompatibility	uint8_t	0	Hardware Compatibility Index
ulHardware Features1	uint32_t	0	Hardware Features 1 (not used, set to 0)
ulHardware Features2	uint32_t	0	Hardware Features 2 (not used, set to 0)
bBootOption	uint8_t	0	Boot Option (not used, set to 0)
bReserved[11]	uint8_t	0	Reserved, set to 0

Table 14: HIL_HW_HARDWARE_INFO_CNF_T – Hardware Info confirmation

Packet structure reference

```
/* READ HARDWARE INFORMATION CONFIRMATION */
#define HIL_HW_HARDWARE_INFO_CNF            HIL_HW_HARDWARE_INFO_REQ+1

typedef struct HIL_HW_HARDWARE_INFO_CNF_DATA_Ttag
{
  uint32_t ulDeviceNumber;         /* device number              */
  uint32_t ulSerialNumber;         /* serial number              */
  uint16_t ausHwOptions[4];        /* hardware assembly options  */
  uint16_t usManufacturer;         /* device manufacturer        */
  uint16_t usProductionDate;       /* production date            */
  uint32_t ulLicenseFlags1;        /* license flags 1            */
  uint32_t ulLicenseFlags2;        /* license flags 2            */
  uint16_t usNetxLicenseID;        /* license ID                 */
  uint16_t usNetxLicenseFlags;     /* license flags              */
  uint16_t usDeviceClass;          /* device class               */
  uint8_t  bHwRevision;            /* hardware revision          */
  uint8_t  bHwCompatibility;       /* hardware compatibility     */
  uint32_t ulHardwareFeatures1;    /* not used, set to 0         */
  uint32_t ulHardwareFeatures2;    /* not used, set to 0         */
  uint8_t  bBootOption;            /* not used, set to 0         */
  uint8_t  bReserved[11];          /* reserved, set to 0         */
} HIL_HW_HARDWARE_INFO_CNF_DATA_T;

typedef struct HIL_HW_HARDWARE_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;  /* packet header     */
  HIL_HW_HARDWARE_INFO_CNF_DATA_T  tData;  /* packet data       */
} HIL_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**
HIL_PACKET_DEST_SYSTEM	0xFFFFFFFF	Version of the operating system
	0 ... 3	Protokol stack name of the communication channel given by *ulChannelID*
HIL_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**
HIL_PACKET_DEST_SYSTEM	0xFFFFFFFF	Version of the operating system
	0 ... 3	Protokol stack name of the communication channel given by *ulChannelID*
HIL_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**
HIL_PACKET_DEST_SYSTEM or HIL_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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000 0x00000020	HIL_PACKET_DEST_SYSTEM HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001EB6	HIL_FIRMWARE_IDENTIFY_REQ
Data			
ulChannelId	uint32_t	see definition above	Channel Identification

Table 15: HIL_FIRMWARE_IDENTIFY_REQ_T – Firmware Identify request

Packet structure reference

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

typedef struct HIL_FIRMWARE_IDENTIFY_REQ_DATA_Ttag
{
  uint32_t   ulChannelId;                   /* channel ID                */
} HIL_FIRMWARE_IDENTIFY_REQ_DATA_T;

typedef struct HIL_FIRMWARE_IDENTIFY_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header             */
  HIL_FIRMWARE_IDENTIFY_REQ_DATA_T tData;   /* packet data               */
} HIL_FIRMWARE_IDENTIFY_REQ_T;
```

Firmware Identify confirmation

The channel firmware returns the following packet.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 76 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EB7	HIL_FIRMWARE_IDENTIFY_CNF
Data			
tFwVersion	Structure	see below	Firmware Version
tFwName	Structure	see below	Firmware Name
tFwDate	Structure	see below	Firmware Date

Table 16: HIL_FIRMWARE_IDENTIFY_CNF_T – Firmware Identify confirmation

Packet structure reference

```
/* IDENTIFY FIRMWARE CONFIRMATION */
#define HIL_FIRMWARE_IDENTIFY_CNF           HIL_FIRMWARE_IDENTIFY_REQ+1

typedef struct HIL_FW_IDENTIFICATION_Ttag
{
  HIL_FW_VERSION_T  tFwVersion;           /* firmware version            */
  HIL_FW_NAME_T     tFwName;              /* firmware name               */
  HIL_FW_DATE_T     tFwDate;              /* firmware date               */
} HIL_FW_IDENTIFICATION_T;

typedef struct HIL_FIRMWARE_IDENTIFY_CNF_DATA_Ttag
{
  HIL_FW_IDENTIFICATION_T   tFirmwareIdentification;    /* firmware ID    */
} HIL_FIRMWARE_IDENTIFY_CNF_DATA_T;

typedef struct HIL_FIRMWARE_IDENTIFY_CNF_Ttag
{
  HIL_PACKET_HEADER                 tHead;  /* packet header             */
  HIL_FIRMWARE_IDENTIFY_CNF_DATA_T  tData;  /* packet data               */
} HIL_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 HIL_FW_VERSION_Ttag
{
  uint16_t              usMajor;            /* major version number */
  uint16_t              usMinor;            /* minor version number */
  uint16_t              usBuild;            /* build number         */
  uint16_t              usRevision;         /* revision number      */
} HIL_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 HIL_FW_NAME_Ttag
{
  uint8_t              bNameLength;        /* length of firmware name    */
  uint8_t              abName[63];         /* firmware name              */
} HIL_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 HIL_FW_DATE_Ttag
{
  uint16_t             usYear;             /* firmware creation year    */
  uint8_t              bMonth;             /* firmware creation month   */
  uint8_t              bDay;               /* firmware creation day     */
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001E32	HIL_SYSTEM_INFORMATION_BLOCK_REQ

Table 17: HIL_READ_SYS_INFO_BLOCK_REQ_T – System Information Block request

Packet structure reference

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

typedef struct HIL_READ_SYS_INFO_BLOCK_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_READ_SYS_INFO_BLOCK_REQ_T;
```

System Information Block confirmation

The following packet is returned.

Variable	Type	Value / Range	Description
ulLen	uint32_t	48 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E33	HIL_SYSTEM_INFORMATION_BLOCK_CNF
Data			
tSystemInfo	Structure		System Information Block See *netX Dual-Port Memory Interface Manual* for more details.

Table 18: HIL_READ_SYS_INFO_BLOCK_CNF_T – System Information Block confirmation

Packet structure reference

```
/* READ SYSTEM INFORMATION BLOCK CONFIRMATION */
#define HIL_SYSTEM_INFORMATION_BLOCK_CNF    HIL_SYSTEM_INFORMATION_BLOCK_REQ+1

typedef struct HIL_READ_SYS_INFO_BLOCK_CNF_DATA_Ttag
{
  NETX_SYSTEM_INFO_BLOCK               tSystemInfo; /* packet data       */
} HIL_READ_SYS_INFO_BLOCK_CNF_DATA_T;

typedef struct HIL_READ_SYS_INFO_BLOCK_CNF_Ttag
{
  HIL_PACKET_HEADER                   tHead;   /* packet header          */
  HIL_READ_SYS_INFO_BLOCK_CNF_DATA_T  tData;   /* packet data            */
} HIL_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*.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001E34	HIL_CHANNEL_INFORMATION_BLOCK_REQ
Data			
ulChannelId	uint32_t	0 … 7	Channel Identifier Port Number, Channel Number

Table 19: HIL_READ_CHNL_INFO_BLOCK_REQ_T – Channel Information Block request

Packet structure reference

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

typedef struct HIL_READ_CHNL_INFO_BLOCK_REQ_DATA_Ttag
  uint32_t ulChannelId;                       /* channel id              */
} HIL_READ_CHNL_INFO_BLOCK_REQ_DATA_T;

typedef struct HIL_READ_CHNL_INFO_BLOCK_REQ_Ttag
{
  HIL_PACKET_HEADER                   tHead;  /* packet header           */
  HIL_READ_CHNL_INFO_BLOCK_REQ_DATA_T tData;  /* packet data             */
} HIL_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.

Variable	Type	Value / Range	Description
`ulLen`	uint32_t	 16 0	Packet Data Length (in Bytes) If `ulSta` = `HIL_S_OK` Otherwise
`ulSta`	uint32_t	See Below	Status / Error Code, see Section 6
`ulCmd`	uint32_t	0x00001E35	HIL_CHANNEL_INFORMATION_BLOCK_CNF
Data			
`tChannelInfo`	Structure		Channel Information Block See *netX Dual-Port Memory Interface Manual* for more details.

Table 20: HIL_READ_CHNL_INFO_BLOCK_CNF_T – Channel Information Block confirmation

Packet structure reference

```
/* READ CHANNEL INFORMATION BLOCK CONFIRMATION */
#define HIL_CHANNEL_INFORMATION_BLOCK_CNF    HIL_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 HIL_READ_CHNL_INFO_BLOCK_CNF_DATA_Ttag
{
  NETX_CHANNEL_INFO_BLOCK           tChannelInfo;  /* channel info block   */
} HIL_READ_CHNL_INFO_BLOCK_CNF_DATA_T;

typedef struct HIL_READ_CHNL_INFO_BLOCK_CNF_Ttag
{
  HIL_PACKET_HEADER                     tHead;     /* packet header        */
  HIL_READ_CHNL_INFO_BLOCK_CNF_DATA_T   tData;     /* packet data          */
} HIL_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 HIL_CHANNEL_TYPE_COMMUNICATION:
    {
      /* This is a communication channel, read an information */
      uint16_t usActualProtocolClass;
      usActualProtocolClass = ChannelInfo->tCom.usProtocolClass;
    }
    break;

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

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

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

    case HIL_CHANNEL_TYPE_UNDEFINED:
    case HIL_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*).

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001E36	HIL_SYSTEM_CONTROL_BLOCK_REQ

Table 21: HIL_READ_SYS_CNTRL_BLOCK_REQ_T – System Control Block request

Packet structure reference

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

typedef struct HIL_READ_SYS_CNTRL_BLOCK_REQ_Ttag
{
  HIL_PACKET_HEADER                 tHead;   /* packet header             */
} HIL_READ_SYS_CNTRL_BLOCK_REQ_T;
```

System Control Block confirmation

The following packet is returned by the firmware.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 8 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E37	HIL_SYSTEM_CONTROL_BLOCK_CNF
Data			
tSystem Control	Structure		System Control Block See *netX Dual-Port Memory Interface Manual* for more details.

Table 22: HIL_READ_SYS_CNTRL_BLOCK_CNF_T – System Control Block confirmation

Packet structure reference

```
/* READ SYSTEM CONTROL BLOCK CONFIRMATION */
#define HIL_SYSTEM_CONTROL_BLOCK_CNF         HIL_SYSTEM_CONTROL_BLOCK_REQ+1

typedef struct HIL_READ_SYS_CNTRL_BLOCK_CNF_DATA_Ttag
{
  NETX_SYSTEM_CONTROL_BLOCK         tSystemControl;
} HIL_READ_SYS_CNTRL_BLOCK_CNF_DATA_T;

typedef struct HIL_READ_SYS_CNTRL_BLOCK_CNF_Ttag
{
  HIL_PACKET_HEADER                     tHead;     /* packet header        */
  HIL_READ_SYS_CNTRL_BLOCK_CNF_DATA_T   tData;     /* packet data          */
} HIL_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*)

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001E38	HIL_SYSTEM_STATUS_BLOCK_REQ

Table 23: HIL_READ_SYS_STATUS_BLOCK_REQ_T – System Status Block request

Packet structure reference

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

typedef struct HIL_READ_SYS_STATUS_BLOCK_REQ_Ttag
{
  HIL_PACKET_HEADER                 tHead;   /* packet header             */
} HIL_READ_SYS_STATUS_BLOCK_REQ_T;
```

System Status Block confirmation

The following packet is returned by the firmware.

Variable	Type	Value / Range	Description
ulLen	uint32_t	64 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E39	HIL_SYSTEM_STATUS_BLOCK_CNF
Data			
tSystemState	Structure		System Status Block See *netX Dual-Port Memory Interface Manual* for more details.

Table 24: HIL_READ_SYS_STATUS_BLOCK_CNF_T – System Status Block confirmation

Packet structure reference

```
/* READ SYSTEM STATUS BLOCK CONFIRMATION */
#define HIL_SYSTEM_STATUS_BLOCK_CNF          HIL_SYSTEM_STATUS_BLOCK_REQ+1

typedef struct HIL_READ_SYS_STATUS_BLOCK_CNF_DATA_Ttag
{
  NETX_SYSTEM_STATUS_BLOCK              tSystemState;
} HIL_READ_SYS_STATUS_BLOCK_CNF_DATA_T;

typedef struct HIL_READ_SYS_STATUS_BLOCK_CNF_Ttag
{
  HIL_PACKET_HEADER                     tHead;     /* packet header        */
  HIL_READ_SYS_STATUS_BLOCK_CNF_DATA_T  tData;     /* packet data          */
} HIL_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 permanently.

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

- set a permanent MAC address if a *Security Memory* is available or
- set a temporary MAC address if no *Security Memory* is available.

Permanently changing the MAC address

- **Hardware without Security Memory / Flash Device Label**
  The MAC address is stored temporarily e.g. the MAC Address is lost after a reset or power cycle. Neither the *Store* flag nor the *Force* flag has a meaning.
- **Hardware with Security Memory**
  - **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.

Note: Storing a MAC address permanently in the Flash Device Label is not supported.

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.

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
`ulLen`	uint32_t	12	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00001EEE	HIL_SET_MAC_ADDR_REQ
Data			
`ulParam`	uint32_t	0x00000001 0x00000002	Parameter Field (see below) HIL_STORE_MAC_ADDRESS HIL_FORCE_MAC_ADDRESS
`abMacAddr[6]`	uint8_t		MAC Address
`abPad[2]`	uint8_t	0x00	Padding bytes, set to zero

Table 25: HIL_SET_MAC_ADDR_REQ_T – Set MAC Address request

Packet structure reference

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

#define HIL_STORE_MAC_ADDRESS               0x00000001
#define HIL_FORCE_MAC_ADDRESS               0x00000002

typedef struct HIL_SET_MAC_ADDR_REQ_DATA_Ttag
{
  uint32_t ulParam;                     /* parameter bit field          */
  uint8_t  abMacAddr[6];                /* MAC address                  */
  uint8_t  abPad[2];                    /* pad bytes, set to zero       */
} HIL_SET_MAC_ADDR_REQ_DATA_T;

typedef struct HIL_SET_MAC_ADDR_REQ_Ttag
{
  HIL_PACKET_HEADER               tHead;       /* packet header          */
  HIL_SET_MAC_ADDR_REQ_DATA_T     tData;       /* packet data            */
} HIL_SET_MAC_ADDR_REQ_T;
```

Parameter ulParam

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

Table 26: Set MAC Address Parameter Field

Bit No.	Definition	Definition / Description
0	*HIL_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* is available.
1	*HIL_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* available.
2 … 31	none	Reserved, set to 0

Table 27: Set MAC Address Parameter

Set MAC Address confirmation

The system channel returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Codes, see Section 6
ulCmd	uint32_t	0x00001EEF	HIL_SET_MAC_ADDR_CNF

Table 28: HIL_SET_MAC_ADDR_CNF_T – Set MAC Address confirmation

Packet structure reference

```
/* SET MAC ADDRESS CONFIRMATION */
#define HIL_SET_MAC_ADDR_CNF                HIL_SET_MAC_ADDR_REQ+1

typedef struct HIL_SET_MAC_ADDR_CNF_Ttag
{
  HIL_PACKET_HEADER               tHead;       /* packet header          */
} HIL_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* = HIL_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 file system of netX 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
	PORT_4	Application Channel 0
	PORT_5	Application Channel 1

Table 29: Folder layout of file system

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

Directory List request

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	6 + n	sizeof(HIL_DIR_LIST_REQ_DATA_T) + strlen("DirName")+1 Remark: 0 can be used for the second, third, etc. packet.
ulCmd	uint32_t	0x00001E70	HIL_DIR_LIST_REQ
ulExt	uint32_t	0x00, 0xC0	0x00: for the first packet. 0xC0: for the next packets.
Data			
ulChannelNo	uint32_t	0 … 3 4 … 5 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 Application Channel 0 … 1 System Channel
usDirName Length	uint16_t	n	Name Length Length of the Directory Name (in Bytes) strlen("DirName")+1
	uint8_t	ASCII	Directory Name ASCII string, zero terminated e.g. "\\PORT_0", "\", etc.

Table 30: HIL_DIR_LIST_REQ_T – Directory List request

Packet structure reference

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

/* Channel Number */
#define HIL_FILE_CHANNEL_0 (0)
#define HIL_FILE_CHANNEL_1 (1)
#define HIL_FILE_CHANNEL_2 (2)
#define HIL_FILE_CHANNEL_3 (3)
#define HIL_FILE_CHANNEL_4 (4)
#define HIL_FILE_CHANNEL_5 (5)
#define HIL_FILE_SYSTEM    (0xFFFFFFFF)

typedef struct HIL_DIR_LIST_REQ_DATA_Ttag
{
  uint32_t  ulChannelNo;        /* 0 = channel 0 ... 5 = channel 5             */
                                /* 0xFFFFFFFF = system, see HIL_FILE_xxxx      */
  uint16_t  usDirNameLength;    /* length of NULL terminated string            */
  /* a NULL-terminated name string will follow here */
} HIL_DIR_LIST_REQ_DATA_T;

typedef struct HIL_DIR_LIST_REQ_Ttag
{
  HIL_PACKET_HEADER           tHead;          /* packet header               */
  HIL_DIR_LIST_REQ_DATA_T     tData;          /* packet data                 */
} HIL_DIR_LIST_REQ_T;
```

Directory List confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	 24 0 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK If ulSta = HIL_S_OK and ulExt = 0x40 (last packet) Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E71	HIL_DIR_LIST_CNF
ulExt	uint32_t	0x80, 0xC0, 0x40	0x80 for the first packet. 0xC0 for the following packets. 0x40 for the last packet.
Data			
szName[16]	uint8_t		File Name
ulFileSize	uint32_t	m	File Size in Bytes
bFileType	uint8_t	 0x00000001 0x00000002	File Type HIL_DIR_LIST_CNF_FILE_TYPE_DIRECTORY HIL_DIR_LIST_CNF_FILE_TYPE_FILE
bReserved	uint8_t	0	Reserved, unused
usReserved2	uint16_t	0	Reserved, unused

Table 31: HIL_DIR_LIST_CBF_T – Directory List confirmation

Packet structure reference

```
/* DIRECTORY LIST CONFIRMATION */
#define HIL_DIR_LIST_CNF                    HIL_DIR_LIST_REQ+1

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

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

typedef struct HIL_DIR_LIST_CNF_DATA_Ttag
{
  uint8_t                 szName[16];    /* file name                    */
  uint32_t                ulFileSize;    /* file size                    */
  uint8_t                 bFileType;     /* file type                    */
  uint8_t                 bReserved;     /* reserved, set to 0           */
  uint16_t                bReserved2     /* reserved, set to 0           */
} HIL_DIR_LIST_CNF_DATA_T;

typedef struct HIL_DIR_LIST_CNF_Ttag
{
  HIL_PACKET_HEADER       tHead;         /* packet header                */
  HIL_DIR_LIST_CNF_DATA_T tData;         /* packet data                  */
} HIL_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 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 and is incremented by one for each new packet. Sequence numbers shall not be skipped or used twice because the netX firmware **cannot** re-assemble file data received out of order.

Note: The HIL_FILE_DOWNLOAD_REQ and HIL_FILE_DOWNLOAD_DATA_REQ commands in the table below show a continues incrementation of the *ulId* field between both commands. This is not required since both commands work independently.

Example:

Single Packet Upload/Download			**Two Packet Upload/Download**		
Definition	**ulId**	**ulExt**	**Definition**	**ulId**	**ulExt**
HIL_FILE_DOWNLOAD_REQ	n	HIL_PACKET_SEQ_NONE	HIL_FILE_DOWNLOAD_REQ	n	HIL_PACKET_SEQ_NONE
HIL_FILE_DOWNLOAD_DATA_REQ	n+1	HIL_PACKET_SEQ_NONE	HIL_FILE_DOWNLOAD_DATA_REQ	n+1	HIL_PACKET_SEQ_FIRST
			HIL_FILE_DOWNLOAD_DATA_REQ	n+2	HIL_PACKET_SEQ_LAST

Multi Packet Upload/Download (split into e.g. 50 data packets)		
Definition	**ulId**	**ulExt**
HIL_FILE_DOWNLOAD_REQ	n	HIL_PACKET_SEQ_NONE
HIL_FILE_DOWNLOAD_DATA_REQ	n+1	HIL_PACKET_SEQ_FIRST
HIL_FILE_DOWNLOAD_DATA_REQ	.....	HIL_PACKET_SEQ_MIDDLE
HIL_FILE_DOWNLOAD_DATA_REQ	n+5	HIL_PACKET_SEQ_MIDDLE
HIL_FILE_DOWNLOAD_DATA_REQ	n+6	HIL_PACKET_SEQ_MIDDLE
HIL_FILE_DOWNLOAD_DATA_REQ	.....	HIL_PACKET_SEQ_MIDDLE
HIL_FILE_DOWNLOAD_DATA_REQ	n+50	HIL_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

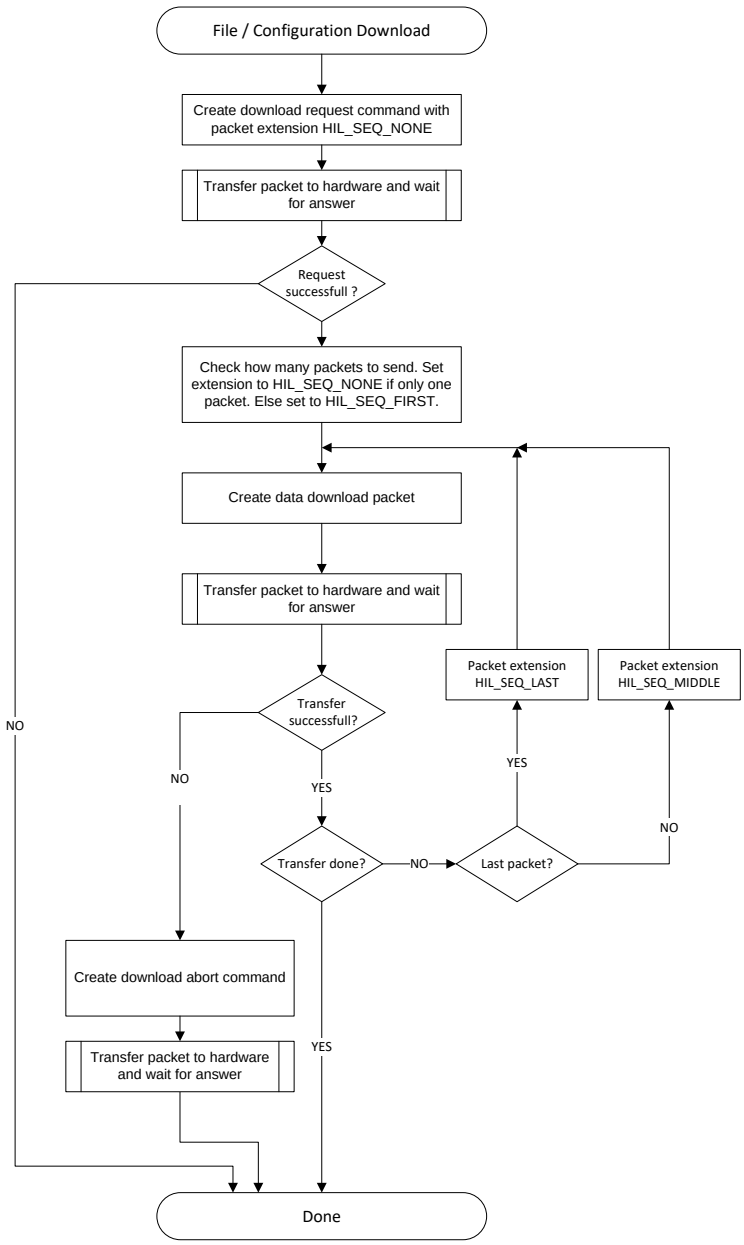


Figure 1: Flowchart File Download

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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_SYSTEM_CHANNEL
ulLen	uint32_t	18 + n	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001E62	HIL_FILE_DOWNLOAD_REQ
ulId	uint32_t	Any	Packet Identifer
ulExt	uint32_t	0x00000000	Extension HIL_PACKET_SEQ_NONE
Data			
ulXferType	uint32_t	0x00000001	Download Transfer Type HIL_FILE_XFER_FILE
ulMaxBlockSize	uint32_t	1 … m	Max Block Size Maximum Size of Block per Packet
ulFileLength	uint32_t	n	File size to be downloaded in bytes
ulChannelNo	uint32_t	0 … 3 4 … 5 0xFFFFFFFF	Destination Channel Number Communication Channel 0 … 3 Application Channel 0 … 1 System Channel
usFileNameLength	uint16_t	n	Length of Name Length of the following file name (in Bytes)
(file name)	uint8_t	ASCII	File Name ASCII string, zero terminated

Table 32: HIL_FILE_DOWNLOAD_REQ_T – File Download request

Packet structure reference

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

/* TRANSFER FILE */
#define HIL_FILE_XFER_FILE                  0x00000001

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

/* TRANSFER MODULE */
#define HIL_FILE_XFER_MODULE                0x00000002

/* Channel Number */
#define HIL_FILE_CHANNEL_0                  (0)
#define HIL_FILE_CHANNEL_1                  (1)
#define HIL_FILE_CHANNEL_2                  (2)
#define HIL_FILE_CHANNEL_3                  (3)
#define HIL_FILE_CHANNEL_4                  (4)
#define HIL_FILE_CHANNEL_5                  (5)
#define HIL_FILE_SYSTEM                     (0xFFFFFFFF)

typedef struct HIL_FILE_DOWNLOAD_REQ_DATA_Ttag
{
  uint32_t ulXferType;
  uint32_t ulMaxBlockSize;
  uint32_t ulFileLength;
  uint32_t ulChannelNo;
  uint16_t usFileNameLength;
  /* a NULL-terminated file name follows here                    */
  /* uint8_t   abFileName[ ];                                    */
} HIL_FILE_DOWNLOAD_REQ_DATA_T;
```

```
typedef struct HIL_FILE_DOWNLOAD_REQ_Ttag
{
  HIL_PACKET_HEADER              tHead;   /* packet header               */
  HIL_FILE_DOWNLOAD_REQ_DATA_T   tData;   /* packet data                 */
} HIL_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.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 4 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E63	HIL_FILE_DOWNLOAD_CNF
ulId	uint32_t	From Request	Packet Identifer
Data			
ulMaxBlockSize	uint32_t	 1 … n	Max Block Size Maximum Size of Block per Packet

Table 33: HIL_FILE_DOWNLOAD_CNF_T – File Download confirmation

Packet structure reference

```
/* FILE DOWNLOAD CONFIRMATION */
#define HIL_FILE_DOWNLOAD_CNF                HIL_FILE_DOWNLOAD_REQ+1

typedef struct HIL_FILE_DOWNLOAD_CNF_DATA_Ttag
{
  uint32_t ulMaxBlockSize;
} HIL_FILE_DOWNLOAD_CNF_DATA_T;

typedef struct HIL_FILE_DOWNLOAD_CNF_Ttag
{
  HIL_PACKET_HEADER              tHead;    /* packet header               */
  HIL_FILE_DOWNLOAD_CNF_DATA_T   tData;    /* packet data                 */
} HIL_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 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 operating system returns an error code in *ulSta*. The user application then has to send an *Abort File Download Request* packet (see page 45) 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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_SYSTEM_CHANNEL
ulLen	uint32_t	8 + n	Packet Data Length (in Bytes)
ulId	uint32_t	id	Packet Identifier **Note:** Should be incremented for each request: ulId (this request) = ulId (previous request) + 1
ulCmd	uint32_t	0x00001E64	HIL_FILE_DOWNLOAD_DATA_REQ
ulExt	uint32_t	0x00000000 0x00000080 0x000000C0 0x00000040	Extension HIL_PACKET_SEQ_NONE (if data fits into one packet) HIL_PACKET_SEQ_FIRST HIL_PACKET_SEQ_MIDDLE HIL_PACKET_SEQ_LAST
Data			
ulBlockNo	uint32_t	0 ... m	Block Number Block or Sequence Number
ulChksum	uint32_t	S	Checksum CRC32 Polynomial
	uint8_t	0 … 0xFF	File Data Block (length given in *ulLen*)

Table 34: HIL_FILE_DOWNLOAD_DATA_REQ_T – File Download Data request

Packet structure reference

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

/* PACKET SEQUENCE */
#define HIL_PACKET_SEQ_NONE                 0x00000000
#define HIL_PACKET_SEQ_FIRST                0x00000080
#define HIL_PACKET_SEQ_MIDDLE               0x000000C0
#define HIL_PACKET_SEQ_LAST                 0x00000040

typedef struct HIL_FILE_DOWNLOAD_DATA_REQ_DATA_Ttag
{
  uint32_t ulBlockNo;               /* block number                      */
  uint32_t ulChksum;                /* cumulative CRC-32 checksum        */
  /* data block follows here                                             */
  /* uint8_t   abData[ ];                                                */
} HIL_FILE_DOWNLOAD_DATA_REQ_DATA_T;
```

```
typedef struct HIL_FILE_DOWNLOAD_DATA_REQ_Ttag
{
  HIL_PACKET_HEADER                  tHead;     /* packet header          */
  HIL_FILE_DOWNLOAD_DATA_REQ_DATA_T  tData;     /* packet data            */
} HIL_FILE_DOWNLOAD_DATA_REQ_T;
```

File Download Data confirmation

The following confirmation packet is returned. It contains the expected CRC32 checksum of the data block.

Variable	Type	Value / Range	Description
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E65	HIL_FILE_DOWNLOAD_DATA_CNF
ulId	uint32_t	From Request	Packet Identifer
Data			
ulExpectedCrc32	uint32_t	S	Checksum Expected CRC32 polynomial Note: If the functions fails, ulExpectedCrc32 contains the latest created CRC32 until detecting an error (could also be 0 if no CRC32 was created at all).

Table 35: HIL_FILE_DOWNLOAD_DATA_CNF_T – File Download Data confirmation

Packet structure reference

```
/* FILE DOWNLOAD DATA CONFIRMATION */
#define HIL_FILE_DOWNLOAD_DATA_CNF          HIL_FILE_DOWNLOAD_DATA_REQ+1

/* PACKET SEQUENCE */
#define HIL_PACKET_SEQ_NONE                 0x00000000

typedef struct HIL_FILE_DOWNLOAD_DATA_CNF_DATA_Ttag
{
  uint32_t ulExpectedCrc32;          /* expected CRC-32 checksum          */
} HIL_FILE_DOWNLOAD_DATA_CNF_DATA_T;

typedef struct HIL_FILE_DOWNLOAD_DATA_CNF_Ttag
{
  HIL_PACKET_HEADER                  tHead;     /* packet header          */
  HIL_FILE_DOWNLOAD_DATA_CNF_DATA_T  tData;     /* packet data            */
} HIL_FILE_DOWNLOAD_DATA_CNF_T;
```

3.8.2.3 File Download Abort

If an error occurs during the download of a file (*ulSta* not equal to HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_SYSTEM_CHANNEL
ulCmd	uint32_t	0x00001E66	HIL_FILE_DOWNLOAD_ABORT_REQ
ulId	uint32_t	Any	Packet Identifer as Unique Number

Table 36: HIL_FILE_DOWNLOAD_ABORT_REQ_T – File Download Abort request

Packet structure reference

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

typedef struct HIL_FILE_DOWNLOAD_ABORT_REQ_Ttag
{
  HIL_PACKET_HEADER     tHead;             /* packet header                */
} HIL_FILE_DOWNLOAD_ABORT_REQ_T;
```

File Download Abort confirmation

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

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E67	HIL_FILE_DOWNLOAD_ABORT_CNF

Table 37: HIL_FILE_DOWNLOAD_ABORT_CNF_T – File Download Abort confirmation

Packet structure reference

```
/* ABORT DOWNLOAD REQUEST */
#define HIL_FILE_DOWNLOAD_ABORT_CNF          HIL_FILE_DOWNLOAD_ABORT_REQ+1

typedef struct HIL_FILE_DOWNLOAD_ABORT_CNF_Ttag
{
  HIL_PACKET_HEADER     tHead;             /* packet header                */
} HIL_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 netX operating system. If the requested file exists, the netX operating system returns all necessary file information in the response.

The host application creates *File Upload Data* request packets, which will be acknowledged by the netX operating system 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

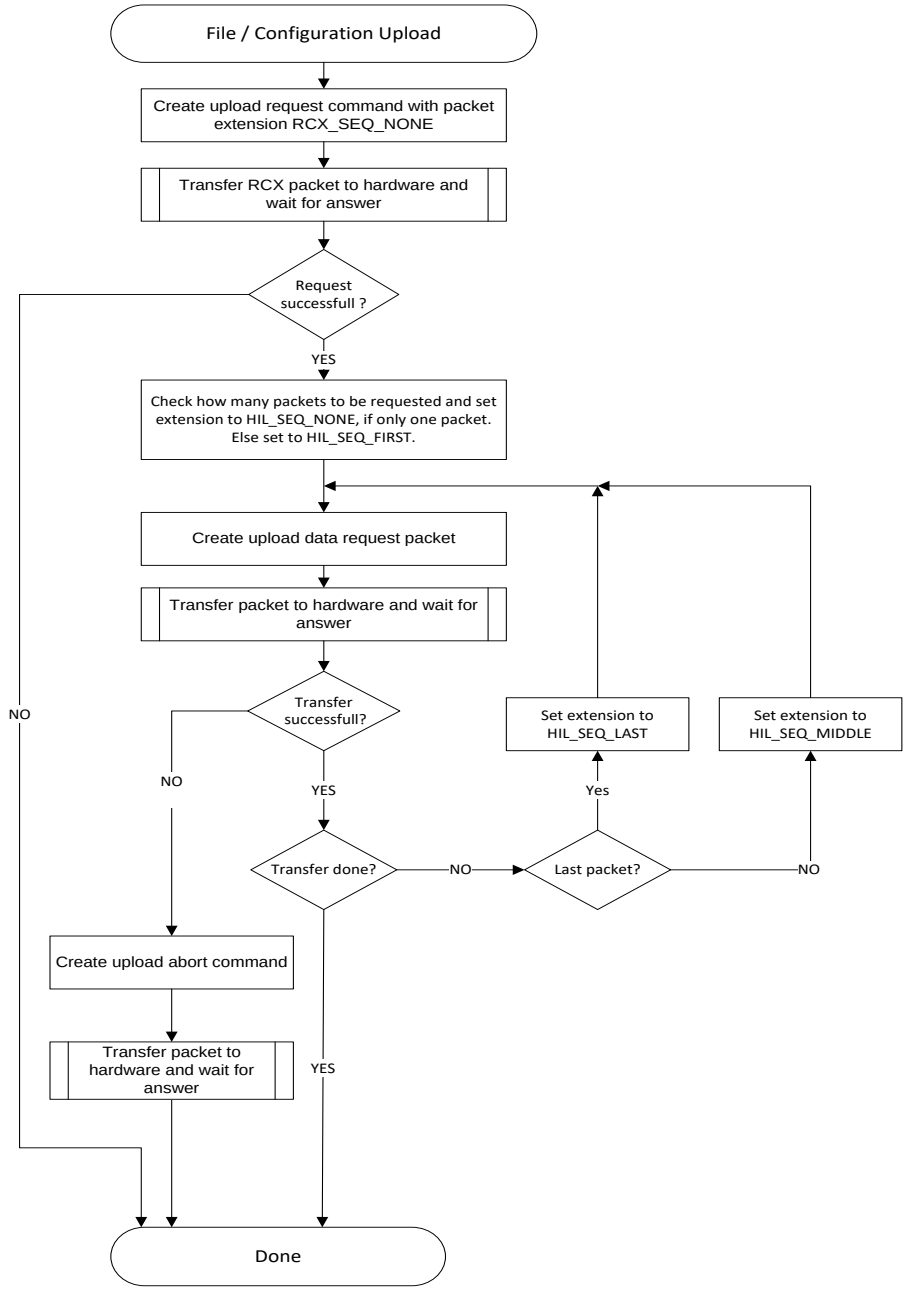


Figure 2: Flowchart File Upload

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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	14 + n	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001E60	HIL_FILE_UPLOAD_REQ
ulId	uint32_t	Any	Packet Identifier
ulExt	uint32_t	0x00000000	Extension HIL_PACKET_SEQ_NONE
Data			
ulXferType	uint32_t	0x00000001	Transfer Type: HIL_FILE_XFER_FILE
ulMaxBlockSize	uint32_t	1 … m	Max Block Size Maximum Size of Block per Packet
ulChannelNo	uint32_t	0 … 3 4 … 5 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 Application Channel 0 … 1 System Channel
usFileNameLength	uint16_t	n	Length of Name Length of Following File Name (in Bytes)
	uint8_t	ASCII	File Name ASCII string, zero terminated

Table 38: HIL_FILE_UPLOAD_REQ_T – File Upload request

Packet structure reference

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

/* PACKET SEQUENCE */
#define HIL_PACKET_SEQ_NONE                0x00000000

/* TRANSFER TYPE */
#define HIL_FILE_XFER_FILE                 0x00000001

/* CHANNEL Number */
#define HIL_FILE_CHANNEL_0                 (0)
#define HIL_FILE_CHANNEL_1                 (1)
#define HIL_FILE_CHANNEL_2                 (2)
#define HIL_FILE_CHANNEL_3                 (3)
#define HIL_FILE_CHANNEL_4                 (4)
#define HIL_FILE_CHANNEL_5                 (5)
#define HIL_FILE_SYSTEM                    (0xFFFFFFFF)
```

```
typedef struct HIL_FILE_UPLOAD_REQ_DATA_Ttag
{
  uint32_t ulXferType;                    /* transfer type               */
  uint32_t ulMaxBlockSize;                /* block size                  */
  uint32_t ulChannelNo;                   /* channel number              */
  uint16_t usFileNameLength;              /* length of file name         */
  /* a NULL-terminated file name follows here                           */
  /* uint8_t   abFileName[ ];                file name                  */
} HIL_FILE_UPLOAD_REQ_DATA_T;

typedef struct HIL_FILE_UPLOAD_REQ_Ttag
{
  HIL_PACKET_HEADER           tHead;      /* packet header               */
  HIL_FILE_UPLOAD_REQ_DATA_T  tData;      /* packet data                 */
} HIL_FILE_UPLOAD_REQ_T;
```

File Upload confirmation

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

Variable	Type	Value / Range	Description
ulLen	uint32_t	 8 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E61	HIL_FILE_UPLOAD_CNF_T
ulId	uint32_t	From Request	Packet Identifier
Data			
ulMaxBlockSize	uint32_t	n	Max Block Size Maximum Size of Block per Packet
ulFileLength	uint32_t	n	File Length Total File Length (in Bytes)

Table 39: HIL_FILE_UPLOAD_CNF_T – File Upload confirmation

Packet structure reference

```
/* FILE UPLOAD CONFIRMATION */
#define HIL_FILE_UPLOAD_CNF                 HIL_FILE_UPLOAD_REQ+1

/* PACKET SEQUENCE */
#define HIL_PACKET_SEQ_NONE                 0x00000000

typedef struct HIL_FILE_UPLOAD_CNF_DATA_Ttag
{
  uint32_t ulMaxBlockSize;            /* maximum block size possible     */
  uint32_t ulFileLength;              /* file size to transfer           */
} HIL_FILE_UPLOAD_CNF_DATA_T;

typedef struct HIL_FILE_UPLOAD_CNF_Ttag
{
  HIL_PACKET_HEADER           tHead;  /* packet header                   */
  HIL_FILE_UPLOAD_CNF_DATA_T  tData;  /* packet data                     */
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001E6E	HIL_FILE_UPLOAD_DATA_REQ
ulId	uint32_t	ulId+1	Packet Identifer **Note:** Should be incremented for each request
ulExt	uint32_t	0x00000000 0x00000080 0x000000C0 0x00000040	Extension HIL_PACKET_SEQ_NONE (if data fits into one packet) HIL_PACKET_SEQ_FIRST HIL_PACKET_SEQ_MIDDLE HIL_PACKET_SEQ_LAST

Table 40: HIL_FILE_UPLOAD_DATA_REQ_T – File Upload Data request

Packet structure reference

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

/* PACKET SEQUENCE */
#define HIL_PACKET_SEQ_NONE                 0x00000000
#define HIL_PACKET_SEQ_FIRST                0x00000080
#define HIL_PACKET_SEQ_MIDDLE               0x000000C0
#define HIL_PACKET_SEQ_LAST                 0x00000040

typedef struct HIL_FILE_UPLOAD_DATA_REQ_Ttag
{
  PACKET_HEADER    tHead;               /* packet header              */
} HIL_FILE_UPLOAD_DATA_REQ_T;
```

File Upload Data confirmation

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

Variable	Type	Value / Range	Description
`ulLen`	uint32_t	8 + n 0	Packet Data Length (in Bytes) If `ulSta` = `HIL_S_OK` Otherwise
`ulSta`	uint32_t	See Below	Status / Error Code, see Section 6
`ulCmd`	uint32_t	0x00001E6F	HIL_FILE_UPLAOD_DATA_CNF
`ulId`	uint32_t	From Request	Packet Identifier
Data			
`ulBlockNo`	uint32_t	0 … m	Block Number Block or Sequence Number
`ulChksum`	uint32_t	S	Checksum CRC32 Polynomial
	uint8_t		File Data Block (Size is n given in *`ulLen`*)

Table 41: HIL_FILE_UPLOAD_DATA_CNF_T – File Upload Data confirmation

Packet structure reference

```
/* FILE DATA UPLOAD CONFIRMATION */
#define HIL_FILE_UPLAOD_DATA_CNF             HIL_FILE_UPLOAD_DATA_REQ +1

/* PACKET SEQUENCE */
#define HIL_PACKET_SEQ_NONE                  0x00000000

typedef struct HIL_FILE_UPLOAD_DATA_CNF_DATA_Ttag
{
  uint32_t ulBlockNo;                     /* block number starting from 0 */
  uint32_t ulChksum;                      /* cumulative CRC-32 checksum   */
  /* data block follows here                                              */
  /* uint8_t   abData[ ];                                                 */
} HIL_FILE_UPLOAD_DATA_CNF_DATA_T;

typedef struct HIL_FILE_UPLOAD_DATA_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header             */
  HIL_FILE_UPLOAD_DATA_CNF_DATA_T  tData;   /* packet data               */
} HIL_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 netX operating system 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 (*ulSta* not equal to HIL_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: *HIL_FILE_UPLOAD_ABORT_REQ_T*			
Variable	**Type**	**Value / Range**	**Description**
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001E5E	HIL_FILE_UPLOAD_ABORT_REQ
ulId	uint32_t	Any	Packet Identifer as Unique Number

Table 42: HIL_FILE_UPLOAD_ABORT_REQ_T – File Upload Abort request

Packet structure reference

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

typedef struct HIL_FILE_UPLOAD_ABORT_REQ_Ttag
{
  HIL_PACKET_HEADER     tHead;            /* packet header               */
} HIL_FILE_UPLOAD_ABORT_REQ_T;
```

File Upload Abort confirmation

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

Structure Information: *HIL_FILE_UPLOAD_ABORT_CNF_T*			
Variable	**Type**	**Value / Range**	**Description**
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E5F	HIL_FILE_UPLOAD_ABORT_CNF

Table 43: HIL_FILE_UPLOAD_ABORT_CNF_T – File Upload Abort confirmation

Packet structure reference

```
/* FILE ABORT UPLOAD CONFIRMATION */
#define HIL_FILE_UPLOAD_ABORT_CNF           HIL_FILE_UPLOAD_ABORT_REQ+1

typedef struct HIL_FILE_UPLOAD_ABORT_CNF_Ttag
{
  HIL_PACKET_HEADER     tHead;            /* packet header               */
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	6 + n	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001E6A	HIL_FILE_DELETE_REQ
Data			
ulChannelNo	uint32_t	0 … 3 4 … 5 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 Application Channel 0 ... 1 System Channel
usFileName Length	uint16_t	n	Length of Name Length of the Following File Name (in Bytes)
	uint8_t	ASCII	File Name ASCII string, zero terminated

Table 44: HIL_FILE_DELETE_REQ_T – File Delete request

Packet structure reference

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

/* Channel Number */
#define HIL_FILE_CHANNEL_0                 (0)
#define HIL_FILE_CHANNEL_1                 (1)
#define HIL_FILE_CHANNEL_2                 (2)
#define HIL_FILE_CHANNEL_3                 (3)
#define HIL_FILE_CHANNEL_4                 (4)
#define HIL_FILE_CHANNEL_5                 (5)
#define HIL_FILE_SYSTEM                    (0xFFFFFFFF)

typedef struct HIL_FILE_DELETE_REQ_DATA_Ttag
{
  uint32_t    ulChannelNo;          /* 0 = channel 0 ... 5 = channel 5        */
                                    /* 0xFFFFFFFF = system, see HIL_FILE_xxxx */
  uint16_t    usFileNameLength;     /* length of NULL-terminated file name    */
  /* a NULL-terminated file name will follow here */
} HIL_FILE_DELETE_REQ_DATA_T;

typedef struct HIL_FILE_DELETE_REQ_Ttag
{
  HIL_PACKET_HEADER            tHead;         /* packet header               */
  HIL_FILE_DELETE_REQ_DATA_T   tData;         /* packet data                 */
} HIL_FILE_DELETE_REQ_T;
```

File Delete confirmation

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E6B	HIL_FILE_DELETE_CNF

Table 45: HIL_FILE_DELETE_CNF_T – File Delete confirmation

Packet structure reference

```
/* FILE DELETE REQUEST */
#define HIL_FILE_DELETE_CNF                  HIL_FILE_DELETE_REQ+1

typedef struct HIL_FILE_DELETE_CNF_Ttag
{
  HIL_PACKET_HEADER       tHead;                /* packet header       */
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	8+m+n	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001E7C	HIL_FILE_RENAME_REQ
Data			
ulChannelNo	uint32_t	0 … 3 4 … 5 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 Application Channel 0 … 1 System Channel
usOldNameLength	uint16_t	m	Length of Old File Name Length of following NULL terminated old File Name (in Bytes)
usNewNameLength	uint16_t	n	Length of New File Name Length of following NULL terminated new File Name (in Bytes)
	uint8_t	ASCII	Old File Name ASCII string, zero terminated
	uint8_t	ASCII	New File Name ASCII string, zero terminated

Table 46: HIL_FILE_RENAME_REQ_T – File Rename request

Packet structure reference

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

#define HIL_FILE_CHANNEL_0                  (0)
#define HIL_FILE_CHANNEL_1                  (1)
#define HIL_FILE_CHANNEL_2                  (2)
#define HIL_FILE_CHANNEL_3                  (3)
#define HIL_FILE_CHANNEL_4                  (4)
#define HIL_FILE_CHANNEL_5                  (5)
#define HIL_FILE_SYSTEM                     (0xFFFFFFFF)

typedef struct HIL_FILE_RENAME_REQ_DATA_Ttag
{
  uint32_t  ulChannelNo;     /* 0..3 = Channel 0..3, 0xFFFFFFFF = System */
  uint16_t  usOldNameLength; /* length of NUL-terminated old file name that will follow
*/
  uint16_t  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 */
} HIL_FILE_RENAME_REQ_DATA_T;

typedef struct HIL_FILE_RENAME_REQ_Ttag
{
  HIL_PACKET_HEADER          tHead;        /* packet header */
  HIL_FILE_RENAME_REQ_DATA_T tData;        /* packet data */
} HIL_FILE_RENAME_REQ_T;
```

File Rename confirmation

Variable	Type	Value / Range	Description
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E7D	HIL_FILE_RENAME_CNF

Table 47: HIL_FILE_RENAME_CNF_T – File Rename confirmation

Packet structure reference

```
/* FILE RENAME CONFIRMATION */
#define HIL_FILE_RENAME_CNF                HIL_FILE_RENAME_REQ+1

typedef struct HIL_FILE_RENAME_CNF_Ttag
{
  HIL_PACKET_HEADER tHead;                 /* packet header */
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	6 + n	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001E68	HIL_FILE_GET_MD5_REQ
Data			
ulChannelNo	uint32_t	0 … 3 4 … 5 0xFFFFFFFF	Channel Number Communication Channel Application Channel 0 … 1 System Channel
usFileName Length	uint16_t	n	Length of Name Length of the Following File Name (in Bytes)
	uint8_t	ASCII	File Name ASCII string, zero terminated

Table 48: HIL_FILE_GET_MD5_REQ_T – File Get MD5 request

Packet structure reference

```
/* REQUEST MD5 FILE CHECKSUM REQUEST */
#define HIL_FILE_GET_MD5_REQ                0x00001E68

#define HIL_FILE_CHANNEL_0                  (0)
#define HIL_FILE_CHANNEL_1                  (1)
#define HIL_FILE_CHANNEL_2                  (2)
#define HIL_FILE_CHANNEL_3                  (3)
#define HIL_FILE_CHANNEL_4                  (4)
#define HIL_FILE_CHANNEL_5                  (5)
#define HIL_FILE_SYSTEM                     (0xFFFFFFFF)

typedef struct HIL_FILE_GET_MD5_REQ_DATA_Ttag
{
  uint32_t   ulChannelNo;         /* 0 = Channel 0 ... 3 = Channel 3,         */
                                  /* 0xFFFFFFFF = System, see HIL_FILE_xxxx   */
  uint16_t   usFileNameLength;    /* length of NULL-terminated file name      */

  /* a NULL-terminated file name will follow here */
} HIL_FILE_GET_MD5_REQ_DATA_T;

typedef struct HIL_FILE_GET_MD5_REQ_Ttag
{
  PACKET_HEADER                   tHead;       /* packet header              */
  HIL_FILE_GET_MD5_REQ_DATA_T     tData;       /* packet data                */
} HIL_FILE_GET_MD5_REQ_T;
```

File Get MD5 confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	 16 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E69	HIL_FILE_GET_MD5_CNF
Data			
abMD5[16]	uint8_t	0 … 0xFF	MD5 checksum

Table 49: HIL_FILE_GET_MD5_CNF_T – File Get MD5 confirmation

Packet structure reference

```
/* REQUEST MD5 FILE CHECKSUM REQUEST */
#define HIL_FILE_GET_MD5_CNF                HIL_FILE_GET_MD5_REQ+1

typedef struct HIL_FILE_GET_MD5_CNF_DATA_Ttag
{
  uint8_t      abMD5[16];          /* MD5 checksum                */
} HIL_FILE_GET_MD5_CNF_DATA_T;

typedef struct HIL_FILE_GET_MD5_CNFtag
{
  HIL_PACKET_HEADER              tHead;     /* packet header       */
  HIL_FILE_GET_MD5_CNF_DATA_T    tData;     /* packet data         */
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	6+n	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001E72	HIL_FILE_GET_HEADER_MD5_REQ
Data			
ulChannelNo	uint32_t	0 … 3 4 … 5 0xFFFFFFFF	Channel Number Communication Channel 0 … 3 Application Channel 0 … 1 System Channel
usFileName Length	uint16_t	n	Length of Name Length of the Following File Name (in Bytes)
	uint8_t	ASCII	File Name ASCII string, zero terminated

Table 50: HIL_FILE_GET_HEADER_MD5_REQ_T – File Get Header MD5 request

Packet structure reference

```
/* REQUEST MD5 FILE HEADER CHECKSUM REQUEST */
#define HIL_FILE_GET_HEADER_MD5_REQ        0x00001E72

/* This packet has the same structure, so we are using a typedef here */
typedef HIL_FILE_GET_MD5_REQ_T  HIL_FILE_GET_HEADER_MD5_REQ_T
```

File Get Header MD5 confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	16 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E73	HIL_FILE_GET_HEADER_MD5_CNF
Data			
abMD5[16]	uint8_t	0 … 0xFF	MD5 checksum

Table 51: HIL_FILE_GET_HEADER_MD5_CNF_T – File Get Header MD5 confirmation

Packet structure reference

```
/* REQUEST MD5 FILE HEADER CHECKSUM CONFIRMATION */
#define HIL_FILE_GET_HEADER_MD5_CNF        HIL_FILE_GET_HEADER_MD5_REQ+1

/* This packet has the same structure, so we are using a typedef here */
typedef HIL_FILE_GET_MD5_CNF_T  HIL_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

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
`ulLen`	uint32_t	8	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00001EF8	HIL_DPM_GET_BLOCK_INFO_REQ
Data			
`ulAreaIndex`	uint32_t	0 … 7	Area Index (see below)
`ulSubblockIndex`	uint32_t	0 … 0xFFFFFFFF	Sub Block Index (see below)

Table 52: HIL_DPM_GET_BLOCK_INFO_REQ_T – DPM Get Block Information request

Packet structure reference

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

typedef struct HIL_DPM_GET_BLOCK_INFO_REQ_DATA_Ttag
{
  uint32_t ulAreaIndex;                     /* area index               */
  uint32_t ulSubblockIndex;                 /* sub block index          */
} HIL_DPM_GET_BLOCK_INFO_REQ_DATA_T;

typedef struct HIL_DPM_GET_BLOCK_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER                 tHead;  /* packet header            */
  HIL_DPM_GET_BLOCK_INFO_REQ_DATA_T tData;  /* packet data              */
} HIL_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.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 28 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EF9	HIL_GET_BLOCK_INFO_CNF
Data			
ulAreaIndex	uint32_t	0, 1, … 7	Area Index (Channel Number)
ulSubblockIndex	uint32_t	0 … 0xFFFFFFFF	Number of Sub Blocks (see below)
ulType	uint32_t	0 … 0x0009	Type of Sub Block (see below)
ulOffset	uint32_t	0 … 0xFFFFFFFF	Offset of Sub Block within the Area
ulSize	uint32_t	0 … 65535	Size of Sub Block (see below)
usFlags	uint16_t	0 … 0x0023	Transmission Flags of Sub Block (see below)
usHandshakeMode	uint16_t	0 … 0x0004	Handshake Mode (see below)
usHandshakeBit	uint16_t	0 … 0x00FF	Bit Position in the Handshake Register
usReserved	uint16_t	0	Reserved, unused

Table 53: HIL_DPM_GET_BLOCK_INFO_CNF_T – DPM Get Block Information confirmation

Packet structure reference

```
/* GET BLOCK INFORMATION CONFIRMATION */
#define HIL_DPM_GET_BLOCK_INFO_CNF          HIL_DPM_GET_BLOCK_INFO_REQ+1

typedef struct HIL_DPM_GET_BLOCK_INFO_CNF_DATA_Ttag
{
  uint32_t ulAreaIndex;      /* area index                                */
  uint32_t ulSubblockIndex;  /* number of sub block                       */
  uint32_t ulType;           /* type of sub block                         */
  uint32_t ulOffset;         /* offset of this sub block within the area  */
  uint32_t ulSize;           /* size of the sub block                     */
  uint16_t usFlags;          /* flags of the sub block                    */
  uint16_t usHandshakeMode;  /* handshake mode                            */
  uint16_t usHandshakeBit;   /* bit position in the handshake register    */
  uint16_t usReserved;       /* reserved                                  */
} HIL_DPM_GET_BLOCK_INFO_CNF_DATA_T;

typedef struct HIL_DPM_GET_BLOCK_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header             */
  HIL_DPM_GET_BLOCK_INFO_CNF_DATA_T tData;   /* packet data               */
} HIL_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 54: 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 55: 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 56: 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 Device Data Provider (DDP)

The *Device Data Provider* (DDP) offeres device-specific data, e.g. serial number, hardware revision or MAC addresses of the device to all components inside a device firmware in a uniform manner. The DDP reads the device data from underlying data sources, e.g. Flash Device Label (FDL) and stores these data in internal RAM. Some of the the data is read-only while other data can be read and written.

Applications can use the *DDP OEM Data Area* (page 68) and *DDP Service Set Data* (see page 72) functions to read and write data. The services using a maximum payload of 80 byte in order to fit into the smallest available mailbox (e.g. the system mailbox) offered by a device.

Writeable data is only stored in a **temporary** buffer and never written back to the underlying data source.

However, the DDP itself has a state that represents, whether or not data is changeable at all. This DDP state is either "***passive***", which allows writing data, or "***active***", which does not allow writing data. Firmware components can register at the DDP and the DDP will inform the components if the state changes.

The following table lists the available data, provided by the DDP service, and shows if the data is only readable or also writeable.

Note: Values marked as ***R/W = Yes***, can be chanded by an application.

Code	System Data Definition	R/W	Data Structure / Data Type
0x00	HIL_DDP_SERVICE_DATATYPE_BASE_DEVICE_DATA	No	HIL_DDP_SERVICE_BASE_DEVICE_DATA_T
0x10	HIL_DDP_SERVICE_DATATYPE_MAC_ADDRESSES_APP	Yes	HIL_DDP_SERVICE_MAC_ADDRESS_T[8] Number of structures = 8
0x20	HIL_DDP_SERVICE_DATATYPE_MAC_ADDRESSES_COM	Yes	HIL_DDP_SERVICE_MAC_ADDRESS_T[4] Number of structuers = 4
0x30	HIL_DDP_SERVICE_DATATYPE_USB_INFORMATION	No	HIL_DDP_SERVICE_USB_INFO_T
0x41 .. 0x4A	HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_0 … HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_9	No	HIL_DDP_SERVICE_LIBSTORAGE_AREA_T
0x51 .. 0x53	HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_CHIP_0 … HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_CHIP_3	No	HIL_DDP_SERVICE_LIBSTORAGE_CHIP_T
Code	**OEM Data Definitions**		
0x60	HIL_DDP_SERVICE_DATATYPE_OEM_OPTIONS	Yes	32-bit value
0x61	HIL_DDP_SERVICE_DATATYPE_OEM_SERIALNUMBER	Yes	String max. 28 including 0 termination
0x62	HIL_DDP_SERVICE_DATATYPE_OEM_ORDERNUMBER	Yes	String max. 32 character including 0 termination
0x63	HIL_DDP_SERVICE_DATATYPE_OEM_HARDWAREREVISION	Yes	String max. 16 character including 0 termination
0x64	HIL_DDP_SERVICE_DATATYPE_OEM_PRODUCTIONDATE	Yes	String max. 32 character including 0 termination
0x65 0x66	HIL_DDP_SERVICE_DATATYPE_OEM_VEDORDATA_0 HIL_DDP_SERVICE_DATATYPE_OEM_VEDORDATA_1	Yes	Byte array, size 80 Byte Byte array, size 32 Byte
Code	**General DDP State**		
0x67	HIL_DDP_SERVICE_DATATYPE_STATE	Yes	32-bit value

Table 57: Device data identification (Device Data Provider)

3.10.1 DDP State Definition

The Device Data Provider supports two states:

- HIL_DDP_SERVICE_STATE_PASSIVE (0)
  - In ***passive*** state, the information marked as "***writeable***" (see *Table 57*), can be changed by an application.
  - The information is not used for the firmware configuration.
  - The `HIL_DDP_SERVICE_GET_REQ` returns error `ERR_HIL_DDP_STATE_INVALID` in the `ulSta` field to inform the requester about the current passive state. However, the requested data is delivered in the confirmation as it is (e.g. the content read from the FDL).
  - The `ulSta` field must be evaluated to verify if the information can be considered as valid.
  - All software components integrated in the firmware will reject component-specific services in ***passive*** DDP state.
- HIL_DDP_SERVICE_STATE_ACTIVE (1)
  - Once, an application switches the state to ***active***, parameters are considered as valid.
  - Parameters are not changeable anymore.
  - Parameters are now usable by the firmware for configuration.
  - A state change is executed by the HIL_DDP_SERVICE_SET_REQ service when setting:
    ulDataType= HIL_DDP_SERVICE_DATATYPE_STATE
    ulValue = HIL_DDP_SERVICE_STATE_ACTIVE
  - The DDP informs all registered firmware components about a state change.
  - The confirmation to the state change request is send back after all registered component functions are processed.
  - Switching back to passive mode is not possible.

Note: A state change from ***passive*** to ***active*** mode may require a certain time until all firmware components recognizes and changes into the new (active) state.
During this time, requests to a firmware component will possibly be answered with `ulSta = ERR_HIL_DDP_STATE_INVALID.` In this case, the application can continue to send the request again, until `ulSta` returns without an error.

Note: The firmware must not be used in the mode ***passive*** if the device is using the firmware to fulfill the requirement "fast bootup time" (e.g. Ethernet/IP QuickConnect or PROFINET FastStartUp).

The current state of the DDP can be read by using the `HIL_DDP_SERVICE_GET_REQ` request with `ulDataType` = `HIL_DDP_SERVICE_DATATYPE_STATE`.

3.10.2 DDP Definitions and Data Structures

The following defines and structures used by the DDP get and set services.

```
#define HIL_PRODUCT_DATA_OEM_IDENTIFICATION_FLAG_SERIALNUMBER_VALID       0x00000001
     /*!< OEM serial number stored in szSerialNumber field is valid */

#define HIL_PRODUCT_DATA_OEM_IDENTIFICATION_FLAG_ORDERNUMBER_VALID        0x00000002
     /*!< OEM order number stored in szOrderNumber is valid */

#define HIL_PRODUCT_DATA_OEM_IDENTIFICATION_FLAG_HARDWAREREVISION_VALID   0x00000004
     /*!< OEM hardware revision stored in szHardwareRevision field is valid */

#define HIL_PRODUCT_DATA_OEM_IDENTIFICATION_FLAG_PRODUCTIONDATA_VALID     0x00000008
     /*!< OEM production date stored in szProductionDate field is valid */

#define HIL_DDP_SERVICE_DATATYPE_BASE_DEVICE_DATA        (0x00)

#define HIL_DDP_SERVICE_DATATYPE_MAC_ADDRESSES_APP       (0x10)

#define HIL_DDP_SERVICE_DATATYPE_MAC_ADDRESSES_COM       (0x20)

#define HIL_DDP_SERVICE_DATATYPE_USB_INFORMATION         (0x30)

#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_0    (0x41)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_1    (0x42)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_2    (0x43)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_3    (0x44)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_4    (0x45)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_5    (0x46)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_6    (0x47)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_7    (0x48)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_8    (0x49)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_9    (0x4A)

#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_CHIP_0    (0x51)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_CHIP_1    (0x52)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_CHIP_2    (0x53)
#define HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_CHIP_3    (0x54)

#define HIL_DDP_SERVICE_DATATYPE_OEM_OPTIONS             (0x60)
#define HIL_DDP_SERVICE_DATATYPE_OEM_SERIALNUMBER        (0x61)
#define HIL_DDP_SERVICE_DATATYPE_OEM_ORDERNUMBER         (0x62)
#define HIL_DDP_SERVICE_DATATYPE_OEM_HARDWAREREVISION    (0x63)
#define HIL_DDP_SERVICE_DATATYPE_OEM_PRODUCTIONDATE      (0x64)
#define HIL_DDP_SERVICE_DATATYPE_OEM_VEDORDATA_0         (0x66) /* 80 Bytes payload */
#define HIL_DDP_SERVICE_DATATYPE_OEM_VEDORDATA_1         (0x67) /* 32 Bytes payload */

#define HIL_DDP_SERVICE_DATATYPE_STATE                   (0x70)

/* DDP number definitions, compare with values in DeviceProductionData.h */

#define HIL_DDP_SERVICE_DEFAULT_NAME_SIZE                (16)

#define HIL_DDP_SERVICE_MAC_APP_NUM                      (4)
#define HIL_DDP_SERVICE_MAC_COM_NUM                      (8)

#define HIL_DDP_SERVICE_FLASH_AREA_NUM                   (10)
#define HIL_DDP_SERVICE_FLASH_CHIP_NUM                   (4)

/* DDP state definitions. */
#define HIL_DDP_SERVICE_STATE_PASSIVE                    (0)
#define HIL_DDP_SERVICE_STATE_ACTIVE                     (1)
```

HIL_DDP_SERVICE_BASE_DEVICE_DATA_T

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_BASE_DEVICE_DATA_Ttag
{
  /* Members as defined in HIL_PRODUCT_DATA_BASIC_DEVICE_DATA_T of DeviceProductionData.h */
  uint16_t  usManufacturer;
  uint16_t  usDeviceClass;
  uint32_t  ulDeviceNumber;
  uint32_t  ulSerialNumber;
  uint8_t   bHwCompatibility;
  uint8_t   bHwRevision;
  uint16_t  usProductionDate;
} HIL_DDP_SERVICE_BASE_DEVICE_DATA_T;
```

HIL_DDP_SERVICE_MAC_ADDRESS_T

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_MAC_ADDRESS_Ttag
{
  uint8_t   abMacAddress[6];
} HIL_DDP_SERVICE_MAC_ADDRESS_T;
```

HIL_DDP_SERVICE_USB_INFO_T

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_USB_INFO_Ttag
{
  uint16_t  usUSBVendorID;                                    /*!< USB Device vendor ID (VID) */
  uint16_t  usUSBProductID;                                   /*!< USB Device product ID (PID) */
  uint8_t   abUSBVendorName[HIL_DDP_SERVICE_DEFAULT_NAME_SIZE]; /*!< USB Vendor name (Byte array) */
  uint8_t   abUSBProductName[HIL_DDP_SERVICE_DEFAULT_NAME_SIZE];/*!< USB Product name (string) */
} HIL_DDP_SERVICE_USB_INFO_T;
```

HIL_DDP_SERVICE_LIBSTORAGE_AREA_T

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_LIBSTORAGE_AREA_Ttag
{
  /* Members as defined in HIL_PRODUCT_DATA_LIBSTORAGE_AREAS_T of DeviceProductionData.h */
  uint32_t  ulContentType;
  uint32_t  ulAreaStart;
  uint32_t  ulAreaSize;
  uint32_t  ulChipNumber;
  int8_t    szName[HIL_DDP_SERVICE_DEFAULT_NAME_SIZE];
  uint8_t   bAccessTyp;
} HIL_DDP_SERVICE_LIBSTORAGE_AREA_T;
```

HIL_DDP_SERVICE_LIBSTORAGE_CHIP_T

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_LIBSTORAGE_CHIP_Ttag
{
  /* Members as defined in HIL_PRODUCT_DATA_LIBSTORAGE_CHIPS_T of DeviceProductionData.h */
  uint32_t  ulChipNumber;
  int8_t    szFlashName[HIL_DDP_SERVICE_DEFAULT_NAME_SIZE];
  uint32_t  ulBlockSize;
  uint32_t  ulFlashSize;
  uint32_t  ulMaxEnduranceCycles;
} HIL_DDP_SERVICE_LIBSTORAGE_CHIP_T;
```

3.10.3 DDP OEM Data Area

The DDP offers a temporary OEM data area, which can be used by device vendors to add their specific information into the device.

The OEM data consists of predefined values (e.g. serial number / order number) and a 112 byte freely usable data area. The only limitation of the data area is that it is only read and writeable in two fix portitions of 80 bytes and 32 byte.

OEM data are also read and writeable by the DDP `HIL_DDP_SERVICE_GET_REQ / HIL_DDP_SERVICE_SET_REQ` services and OEM data specific data type definitions (see Table 58) are available to select the data in the services.

Like other DDP data, the OEM data also have a fix data size corresponding to the data type, also defined in Table 59.

Definition of HIL_DDP_SERVICE_DATATYPE_OEM_OPTIONS

This value has a special meaning in the OEM data and the DDP evaluates this value too.

The lowest 4 bits (3:0) of the parameter are intended to ensure that the four predefined OEM parameters belonging together and are inherently consistent. This is necessary because each parameter can individually be written but only useable in conjunction with the OEM other parameters.

Note:	The DDP only accepts the bits (3:0) either cleared or set in the OEM option value. Any attempts to write another value for these bits will be reject with `ulSta` set to `HIL_ERROR_INVALID_DDP_CONTENT` in the confirmation to the request. Other bits are not evaluated.

- HIL_DDP_SERVICE_DATATYPE_OEM_OPTIONS (0x0 / bits (3:0) = 0)
  - OEM parameters are **not valid** / **not consistent**
    HIL_DDP_SERVICE_DATATYPE_OEM_SERIALNUMBER
    HIL_DDP_SERVICE_DATATYPE_OEM_ORDERNUMBER
    HIL_DDP_SERVICE_DATATYPE_OEM_HARDWAREREVISION
    HIL_DDP_SERVICE_DATATYPE_OEM_PRODUCTIONDATE
  - The firmware evaluates the OEM option value and does not use these values.
- HIL_DDP_SERVICE_DATATYPE_OEM_OPTIONS (0xF / bits (3:0) = 1)
  - OEM parameters are **valid** / **consistent**
    HIL_DDP_SERVICE_DATATYPE_OEM_SERIALNUMBER
    HIL_DDP_SERVICE_DATATYPE_OEM_ORDERNUMBER
    HIL_DDP_SERVICE_DATATYPE_OEM_HARDWAREREVISION
    HIL_DDP_SERVICE_DATATYPE_OEM_PRODUCTIONDATE
  - An OEM application hast to set the value to 0xF before the firmware will use the values.

3.10.4 DDP Service Get Data

The application can use this service to read device data from the Device Data Provider (DDP).

DDP Service Get Data Request

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001EEA	HIL_DDP_SERVICE_GET_REQ
Data			
ulDataType	uint32_t	see Table 60	One of the HIL_DDP_SERVICE_DATATYPE_* defines described above.

Table 61: HIL_DDP_SERVICE_GET_REQ_T – Device Data Provider Get request

Packet structure reference

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_GET_REQ_DATA_Ttag
{
  uint32_t                          ulDataType; /*!< DDP_SERVICE_DATATYPE_* definitions */
} HIL_DDP_SERVICE_GET_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_GET_REQ_Ttag
{
  HIL_PACKET_HEADER_T               tHead;      /*!< packet header */
  HIL_DDP_SERVICE_GET_REQ_DATA_T    tData;      /*!< packet data */
} HIL_DDP_SERVICE_GET_REQ_T;
```

DDP Service Get Data Confirmation

Changeable parameter (see *Table 57*) will return ERR_HIL_DDP_STATE_INVALID in ulSta and the corresponding data if the DDP is in state ***passive***.

Variable	Type	Value / Range	Description
ulLen	uint32_t	4 + n 4 + n 0	Packet Data Length (in Bytes / see example below)) If ulSta = SUCCESS_HIL_OK If ulSta = ERR_HIL_DDP_STATE_INVALID Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EEB	HIL_DDP_SERVICE_GET_CNF
Data			
ulDataType	uint32_t	from request	The HIL_DDP_SERVICE_DATATYPE_* define sent in the request packet.
uDataType	see ulDataType	0..n	Data corresponding to the requested data type (see examples below)

Table 62: HIL_DDP_SERVICE_GET_CNF_T – Device Data Provider Get confirmation

Note: The confirmation packet length (ulLen) depends on the requested / returned data type (ulDataType).

Example on how to determine the confirmation data length (n)

```
ulDataType = HIL_DDP_SERVICE_DATATYPE_BASE_DEVICE_DATA
n          = sizeof(HIL_DDP_SERVICE_BASE_DEVICE_DATA_T
ulLen      = 4 + n
```

```
ulDataType = HIL_DDP_SERVICE_DATATYPE_MAC_ADDRESSES_APP
n          = sizeof(HIL_DDP_SERVICE_MAC_ADDRESS_T) * HIL_DDP_SERVICE_MAC_APP_NUM
ulLen      = 4 + n
```

```
ulDataType = HIL_DDP_SERVICE_DATATYPE_OEM_VENDORDATA_0
n          = 80 (Byte array of 80 Byte)
ulLen      = 4 + n
```

```
ulDataType = HIL_DDP_SERVICE_DATATYPE_OEM_VENDORDATA_1
n          = 32 (Byte array of 32 Byte)
ulLen      = 4 + n
```

Packet structure reference

HIL_DDP_SERVICE_GET_CNF_T

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_GET_CNF_Ttag
{
  HIL_PACKET_HEADER_T                 tHead;       /*!< packet header */
  HIL_DDP_SERVICE_GET_CNF_DATA_T      tData;       /*!< packet data */
} HIL_DDP_SERVICE_GET_CNF_T;
```

HIL_DDP_SERVICE_GET_CNF_DATA_T

Note: The confrmation data structure contains the `ulDataType` return value and the resulting data structured in the `uDataType` union.
The union should help to simplify the access to the necessary data depending on the data type.

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_GET_CNF_DATA_Ttag
{
  /*!< DDP_SERVICE_DATATYPE_* definitions */
  uint32_t ulDataType;

  union HIL_DDP_SERVICE_GET_DATATYPE_U
  {
    /* Fixed structures for specific ulDataType */
    HIL_DDP_SERVICE_BASE_DEVICE_DATA_T    tBaseDeviceData;

    /*!< HIL_DDP_SERVICE_DATATYPE_USB_INFORMATION */
    HIL_DDP_SERVICE_USB_INFO_T            tUSBInfo;

   /*!< HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_AREA_* */
    HIL_DDP_SERVICE_LIBSTORAGE_AREA_T     tFlashArea;

    /*!< HIL_DDP_SERVICE_DATATYPE_STORAGE_FLASH_CHIP_* */
    HIL_DDP_SERVICE_LIBSTORAGE_CHIP_T     tFlashChip;

    /* Members for multiple keys (ulDataType) */

    /*!< Keys with 32bit values, e.g. OEM Option Flags */
    uint32_t                              ulValue;

    /* The following values are defined with maximum value sizes.
       Actual valid or used length/sizes dependent on DataType and maybe smaller.*/

    /*!< Strings, e.g. OEM Serial Number; maximum 80 bytes (including NULL termination) */
    int8_t                                szString[80];

    /*!< Binary data, e.g. OEM Vendor Data; maximum 80 bytes */
    uint8_t                              abData[80];

    /*!< HIL_DDP_SERVICE_DATATYPE_MAC_ADDRESSES_*; maximum 13 addresses (6bytes*13=78) */
    HIL_DDP_SERVICE_MAC_ADDRESS_T         atMacAddress[13];

  } uDataType;

} HIL_DDP_SERVICE_GET_CNF_DATA_T;
```

3.10.5 DDP Service Set Data

The application can use this service to change the writable device data in the DDP (Table 63) before configuring the firmware to setup any necessary information for the field bus protocol, the hardware or OEM data.

Once the device data is set, the application has to switch the DDP state to **active**.

This is done by sending HIL_DDP_SERVICE_SET_REQ with
`ulDataType` set to `HIL_DDP_SERVICE_DATATYPE_STATE` and the data value
`HIL_DDP_SERVICE_STATE_ACTIVE`
The firmware will consider the information valid and configure its resources.

Note: Changed device data is only stored in a **temporary** buffer and not written to the underlying data source (e.g. Flash Device Label).
After a system reset or power cycle, the application must repeat this service.

DDP Service Set Request

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
`ulLen`	uint32_t	4 + n	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00001EEC	HIL_DDP_SERVICE_SET_REQ
Data			
`ulDataType`	uint32_t	see Table 64	One of the HIL_DDP_SERVICE_DATATYPE_* defines described above.
`uDataType`	see ulDataType	0..n	Data, depending on `ulDataType`

Table 65: HIL_DDP_SERVICE_SET_REQ_T – Device Data Provider Set request

HIL_DDP_SERVICE_SET_REQ_T

```
/* Set data structure equal to data structure of Get confirmation */
typedef HIL_DDP_SERVICE_GET_CNF_DATA_T HIL_DDP_SERVICE_SET_REQ_DATA_T;

typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_SET_REQ_Ttag
{
  HIL_PACKET_HEADER_T                 tHead;       /*!< packet header */
  HIL_DDP_SERVICE_SET_REQ_DATA_T      tData;       /*!< packet data */
} HIL_DDP_SERVICE_SET_REQ_T;
```

DDP Service Set Data Confirmation

Variable	Type	Value / Range	Description
`ulSta`	uint32_t	See below	Status / Error Code, see Section 6
`ulCmd`	uint32_t	0x00001EED	HIL_DDP_SERVICE_SET_CNF

Table 66: HIL_DDP_SERVICE_SET_CNF_T – Device Data Provider Set confirmation

DDP SERVICE SET CONFIRMATION

```
typedef __HIL_PACKED_PRE struct __HIL_PACKED_POST HIL_DDP_SERVICE_SET_CNF_Ttag
{
  HIL_PACKET_HEADER_T                 tHead;       /*!< packet header */
} HIL_DDP_SERVICE_SET_CNF_T;
/* DDP SERVICE SET CONFIRMATION */
```

3.11 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 75 for details).

3.11.1 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_t	MacAddress[6]	Ethernet Medium Access Address, 6 Bytes
0x06	uint32_t	SdramGeneralCtrl	SDRAM control register value
0x0A	uint32_t	SdramTimingCtrl	SDRAM timing register value
0x0E	uint8_t	SdramSizeExp	SDRAM size in Mbytes
0x0F	uint16_t	HwOptions[4]	Hardware Assembly Option, 4 Words
0x17	uint8_t	BootOption	Boot Option
0x18	uint8_t	Reserved[6]	Reserved, 6 Bytes
0x1E	uint8_t	Zone1Revision	Revision Structure of Zone 1
0x1F	uint8_t	Zone1Checksum	Checksum of Byte 0 to 30

Table 67: Hardware Configuration (Zone 1)

Zone 2 – PCI System and OS Settings

Offset	Type	Name	Description
0x00	uint16_t	PciVendorID	PCI Settings
0x02	uint16_t	PciDeviceID	
0x04	uint8_t	PciSubClassCode	
0x05	uint8_t	PciClassCode	
0x06	uint16_t	PciSubsystemVendorID	
0x08	uint16_t	PciSubsystemDeviceID	
0x0A	uint8_t	PciSizeTarget[3]	
0x0D	uint8_t	PciSizeIO	
0x0E	uint8_t	PciSizeROM[3]	
0x11	uint8_t	Reserved	
0x12	uint8_t	OsSettings[12]	OS Related Information, 12 Bytes
0x1E	uint8_t	Zone2Revision	Revision Structure of Zone 2
0x1F	uint8_t	Zone2Checksum	Checksum of Byte 0 to 30

Table 68: PCI System and OS Setting (Zone 2)

Zone 3 – User Specific Zone

Offset	Type	Name	Description
0 – 0x1F	uint8_t	UserSpecific[32]	Reserved, 32 Byte

Table 69: User Specific Zone (Zone 3)

Memory Zones Structure Reference

```
typedef struct HIL_SECURITY_MEMORY_ZONE1tag
{
  uint8_t  MacAddress[6];          /* Ethernet medium access address      */
  uint32_t SdramGeneralCtrl;       /* SDRAM control register value        */
  uint32_t SdramTimingCtrl;        /* SDRAM timing register value         */
  uint8_t  SdramSizeExp;           /* SDRAM size in Mbytes                */
  uint16_t HwOptions[4];           /* hardware assembly option            */
  uint8_t  BootOption;             /* boot option                         */
  uint8_t  Reserved[6];            /* reserved (6 bytes)                  */
  uint8_t  Zone1Revision;          /* revision structure of zone 1        */
  uint8_t  Zone1Checksum;          /* checksum of byte 0 to 30            */
} HIL_SECURITY_MEMORY_ZONE1;

typedef struct HIL_SECURITY_MEMORY_ZONE2tag
{
  uint16_t PciVendorID;            /* PCI settings                        */
  uint16_t PciDeviceID;            /* PCI settings                        */
  uint8_t  PciSubClassCode;        /* PCI settings                        */
  uint8_t  PciClassCode;           /* PCI settings                        */
  uint16_t PciSubsystemVendorID;   /* PCI settings                        */
  uint16_t PciSubsystemDeviceID;   /* PCI settings                        */
  uint8_t  PciSizeTarget[3];       /* PCI settings                        */
  uint8_t  PciSizeIO;              /* PCI settings                        */
  uint8_t  PciSizeROM[3];          /* PCI settings                        */
  uint8_t  Reserved;
  uint8_t  OsSettings[12];         /* OS Related Information              */
  uint8_t  Zone2Revision;          /* Revision Structure of Zone 2        */
  uint8_t  Zone2Checksum;          /* Checksum of Byte 0 to 30            */
} HIL_SECURITY_MEMORY_ZONE2;

typedef struct HIL_SECURITY_MEMORY_ZONE3tag
{
  uint8_t  UserSpecific[32];       /* user specific area                  */
} HIL_SECURITY_MEMORY_ZONE3;
```

3.11.2 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.

3.11.3 Zone Read

Security Memory Read request

Read information from the Security Memory (if available).

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001EBC	HIL_SECURITY_EEPROM_READ_REQ
Data			
ulZoneId	uint32_t	0x00000001 0x00000002 0x00000003	Zone Identifier HIL_SECURITY_EEPROM_ZONE_1 HIL_SECURITY_EEPROM_ZONE_2 HIL_SECURITY_EEPROM_ZONE_3

Table 70: HIL_SECURITY_EEPROM_READ_REQ_T – Security Memory Read request

Packet structure reference

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

/* Memory Zones */
#define HIL_SECURITY_EEPROM_ZONE_1           0x00000001
#define HIL_SECURITY_EEPROM_ZONE_2           0x00000002
#define HIL_SECURITY_EEPROM_ZONE_3           0x00000003

typedef struct HIL_SECURITY_EEPROM_READ_REQ_DATA_Ttag
{
  uint32_t ulZoneId;                        /* zone identifier             */
} HIL_SECURITY_EEPROM_READ_REQ_DATA_T;

typedef struct HIL_SECURITY_EEPROM_READ_REQ_Ttag
{
  HIL_PACKET_HEADER                     tHead;    /* packet header       */
  HIL_SECURITY_EEPROM_READ_REQ_DATA_T   tData;    /* packet data         */
} HIL_SECURITY_EEPROM_READ_REQ_T;
```

Security Memory Read confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	 32 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EBD	HIL_SECURITY_EEPROM_READ_CNF
Data			
abZoneData[32]	uint8_t	0 … 0xFF	Data from requested zone (size is 32 Byte)

Table 71: HIL_SECURITY_EEPROM_READ_CNF_T – Security Memory Read confirmation

Packet structure reference

```
/* READ SECURITY EEPROM CONFIRMATION */
#define HIL_SECURITY_EEPROM_READ_CNF         HIL_SECURITY_EEPROM_READ_REQ+1

typedef struct HIL_SECURITY_EEPROM_READ_CNF_DATA_Ttag
{
  uint8_t  abZoneData[32];                    /* zone data               */
} HIL_SECURITY_EEPROM_READ_CNF_DATA_T;

typedef struct HIL_SECURITY_EEPROM_READ_CNF_Ttag
{
  HIL_PACKET_HEADER                   tHead;  /* packet header           */
  HIL_SECURITY_EEPROM_READ_CNF_DATA_T tData;  /* packet data             */
} HIL_SECURITY_EEPROM_READ_CNF_T;
```

3.11.4 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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	36	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001EBE	HIL_SECURITY_EEPROM_WRITE_REQ
Data			
ulZoneId	uint32_t	0x00000001 0x00000002 0x00000003	Zone Identifier HIL_SECURITY_EEPROM_ZONE_1 HIL_SECURITY_EEPROM_ZONE_2 HIL_SECURITY_EEPROM_ZONE_3
abZoneData[32]	uint8_t	0 ... 0xFF	Zone data (size is 32 byte)

Table 72: HIL_SECURITY_EEPROM_WRITE_REQ_T – Security Memory Write request

Packet structure reference

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

/* Memory Zones */
#define HIL_SECURITY_EEPROM_ZONE_1           0x00000001
#define HIL_SECURITY_EEPROM_ZONE_2           0x00000002
#define HIL_SECURITY_EEPROM_ZONE_3           0x00000003

typedef struct HIL_SECURITY_EEPROM_WRITE_REQ_DATA_Ttag
{
  uint32_t ulZoneId;          /* zone ID, see HIL_SECURITY_EEPROM_ZONE_x   */
  uint8_t  abZoneData[32];    /* zone data                                 */
} HIL_SECURITY_EEPROM_WRITE_REQ_DATA_T;

typedef struct HIL_SECURITY_EEPROM_WRITE_REQ_Ttag
{
  HIL_PACKET_HEADER                     tHead;   /* packet header         */
  HIL_SECURITY_EEPROM_WRITE_REQ_DATA_T  tData;   /* packet data           */
} HIL_SECURITY_EEPROM_WRITE_REQ_T;
```

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

Security Memory Write confirmation

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EBF	HIL_SECURITY_EEPROM_WRITE_CNF

Table 73: HIL_SECURITY_EEPROM_WRITE_CNF_T – Security Memory Write confirmation

Packet structure reference

```
/* WRITE SECURITY EEPROM CONFIRMATION */
#define HIL_SECURITY_EEPROM_WRITE_CNF        HIL_SECURITY_EEPROM_WRITE_REQ+1

typedef struct HIL_SECURITY_EEPROM_WRITE_CNF_Ttag
{
  HIL_PACKET_HEADER       tHead;          /* packet header               */
} HIL_SECURITY_EEPROM_WRITE_CNF_T;
```

3.12 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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulCmd	uint32_t	0x00001EF4	HIL_HW_LICENSE_INFO_REQ

Table 74: HIL_HW_LICENSE_INFO_REQ_T – HW Read License request

Packet structure reference

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

typedef struct HIL_HW_LICENSE_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER     tHead;            /* packet header               */
} HIL_HW_LICENSE_INFO_REQ_T;
```

HW Read License confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	12 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EF5	HIL_HW_LICENSE_INFO_CNF
Data			
ulLicenseFlags1	uint32_t	0 … 0xFFFFFFFF	License Flags 1
ulLicenseFlags2	uint32_t	0 … 0xFFFFFFFF	License Flags 2
usNetxLicenseID	uint16_t	0 … 0xFFFF	netX License Identification
usNetxLicenseFlags	uint16_t	0 … 0xFFFF	netX License Flags

Table 75: HIL_HW_LICENSE_INFO_CNF_T – HW Read License confirmation

Packet structure reference

```
/* OBTAIN LICENSE INFORMATION CONFIRMATION */
#define HIL_HW_LICENSE_INFO_CNF              HIL_HW_LICENSE_INFO_REQ+1

typedef struct HIL_HW_LICENSE_INFO_CNF_DATA_Ttag
{
  uint32_t ulLicenseFlags1;        /* License Flags 1        */
  uint32_t ulLicenseFlags2;        /* License Flags 2        */
  uint16_t usNetxLicenseID;        /* License ID             */
  uint16_t usNetxLicenseFlags;     /* License Flags          */
} HIL_HW_LICENSE_INFO_CNF_DATA_T;

typedef struct HIL_HW_LICENSE_INFO_CNFtag
{
  HIL_PACKET_HEADER                tHead;  /* packet header              */
  HIL_HW_LICENSE_INFO_CNF_DATA_T   tData;  /* packet data                */
} HIL_HW_LICENSE_INFO_CNF_T;
```

3.13 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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001ED4	HIL_GET_PERF_COUNTERS_REQ
Data			
usStartToken	uint16_t	0 … 0xFFFF	Start token of values
usTokenCount	uint16_t	0 … 0xFFFF	Number of tokens

Table 76: HIL_GET_PERF_COUNTERS_REQ_T – Get Perf Counters request

Packet structure reference

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

typedef struct HIL_GET_PERF_COUNTERS_REQ_DATA_Ttag
{
  uint16_t usStartToken;
  uint16_t usTokenCount;
} HIL_GET_PERF_COUNTERS_REQ_DATA_T;

typedef struct HIL_GET_PERF_COUNTERS_REQ_Ttag
{
  HIL_PACKET_HEADER                      tHead;   /* packet header      */
  HIL_GET_PERF_COUNTERS_REQ_DATA_T       tData;   /* packet data        */
} HIL_GET_PERF_COUNTERS_REQ_T;
```

Get Perf Counters confirmation

The following packet is returned by the firmware.

Variable	Type	Value / Range	Description
ulLen	uint32_t	4 + 8 + (n * 8) 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001ED5	HIL_GET_PERF_COUNTERS_CNF
Data			
usStartToken	uint16_t	from Request	Start token given in the request
usTokenCount	uint16_t	n	Max. number of token in the following array
tPerfSystemUptime	Structure		System up time
atPerfCounters[1]	Structure		Counters

Table 77: HIL_GET_PERF_COUNTERS_CNF_T – Get Perf Counters confirmation

Packet structure reference

```
/* READ PERFORMANCE COUNTER CONFIRMATION */
#define HIL_GET_PERF_COUNTERS_CNF           HIL_GET_PERF_COUNTERS_REQ+1

typedef struct HIL_PERF_COUNTER_DATA_Ttag
{
  uint32_t ulNanosecondsLower;
  uint32_t ulNanosecondsUpper;
} HIL_PERF_COUNTER_DATA_T;

typedef struct HIL_GET_PERF_COUNTERS_CNF_DATA_Ttag
{
  uint16_t                 usStartToken;
  uint16_t                 usTokenCount;
  HIL_PERF_COUNTER_DATA_T  tPerfSystemUptime;
/*  dynamic array, length is given indirectly by ulLen                  */
  HIL_PERF_COUNTER_DATA_T  atPerfCounters[1];
} HIL_GET_PERF_COUNTERS_CNF_DATA_T;

typedef struct HIL_GET_PERF_COUNTERS_CNF_Ttag
{
  HIL_PACKET_HEADER                       tHead;   /* packet header      */
  HIL_GET_PERF_COUNTERS_CNF_DATA_T        tData;   /* packet data        */
} HIL_GET_PERF_COUNTERS_CNF_T;
```

3.14 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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	12	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001ED8	HIL_TIME_COMMAND_REQ
Data			
ulTimeCmd	uint32_t	0x00000001 0x00000002 0x00000003	Time Command TIME_CMD_GETSTATE TIME_CMD_GETTIME TIME_CMD_SETTIME
ulData	uint32_t	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_t	0	Reserved, set to 0

Table 78: HIL_TIME_CMD_REQ_T – Time Command request

Packet structure reference

```
/* Time Packet Command */
#define HIL_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 HIL_TIME_CMD_DATA_Ttag
{
  uint32_t ulTimeCmd;                     /* time command                */
  uint32_t ulData;                        /* data, corresponds to command */
  uint32_t ulReserved;                    /* Reserved                    */
} HIL_TIME_CMD_DATA_T;

typedef struct HIL_TIME_CMD_REQ_Ttag
{
  HIL_PACKET_HEADER       tHead;          /* packet header               */
  HIL_TIME_CMD_DATA_T     tData;          /* packet data                 */
} HIL_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 79: 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.

Variable	Type	Value / Range	Description
ulLen	uint32_t	12 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001ED9	HIL_TIME_COMMAND_CNF
Data			
ulTimeCmd	uint32_t	0x00000001 0x00000002 0x00000003	Time Commands TIME_CMD_GETSTATE TIME_CMD_GETTIME TIME_CMD_SETTIME
ulData	uint32_t	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_t	0	Reserved, set to 0

Table 80: HIL_TIME_CMD_CNF_T – Time Command confirmation

Packet structure reference

```
/* Time Packet Command */
#define HIL_TIME_COMMAND_CNF              HIL_TIME_COMMAND_REQ+1

typedef struct HIL_TIME_CMD_DATA_Ttag
{
  uint32_t ulTimeCmd;                     /* time command                */
  uint32_t ulData;                        /* corresponds to command      */
  uint32_t ulReserved;                    /* reserved                    */
} HIL_TIME_CMD_DATA_T;

typedef struct HIL_TIME_CMD_CNF_Ttag
{
  HIL_PACKET_HEADER       tHead;          /* packet header               */
  HIL_TIME_CMD_DATA_T     tData;          /* packet data                 */
} HIL_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 81: 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 82: 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.

3.15 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* is used to select the channel on which the firmware has to be started.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001EC4	HIL_CHANNEL_INSTANTIATE_REQ
Data			
ulChannelNo	uint32_t	0..3	Communication Channel Number

Table 83: HIL_CHANNEL_INSTANTIATE_REQ_T – Start Firmware request

Packet structure reference

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

typedef struct HIL_CHANNEL_INSTANTIATE_REQ_DATA_Ttag
{
  uint32_t                     ulChannelNo;         /* channel number    */
} HIL_CHANNEL_INSTANTIATE_REQ_DATA_T;

typedef struct HIL_CHANNEL_INSTANTIATE_REQ_Ttag
{
  HIL_PACKET_HEADER                    tHead;       /* packet header     */
  HIL_CHANNEL_INSTANTIATE_REQ_DATA_T   tData;       /* packet data       */
} HIL_CHANNEL_INSTANTIATE_REQ_T;
```

Start Firmware confirmation

The system channel returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EC5	HIL_CHANNEL_INSTANTIATE_CNF

Table 84: HIL_CHANNEL_INSTANTIATE_CNF_T – Start Firmware confirmation

Packet structure reference

```
/* INSTANTIATE FIRMWARE CONFIRMATION */
#define HIL_CHANNEL_INSTANTIATE_CNF          HIL_CHANNEL_INSTANTIATE_REQ+1

typedef struct HIL_CHANNEL_INSTANTIATE_CNF_Ttag
{
  HIL_PACKET_HEADER               tHead;            /* packet header     */
} HIL_CHANNEL_INSTANTIATE_CNF_T;
```

3.16 Second Stage Bootloader

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

3.16.1 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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
ulLen	uint32_t	8	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00001ED6	HIL_FORMAT_REQ
Data			
ulFlags	uint32_t	0x00000000 0x00000001	Type of format operation HIL_FORMAT_REQ_DATA_FLAGS_QUICKFORMAT HIL_FORMAT_REQ_DATA_FLAGS_FULLFORMAT
ulReserved	uint32_t	0	Reserved, unsed

Table 85: HIL_FORMAT_REQ_T – Format request

Packet structure reference

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

#define HIL_FORMAT_REQ_DATA_FLAGS_QUICKFORMAT 0x00000000
#define HIL_FORMAT_REQ_DATA_FLAGS_FULLFORMAT  0x00000001

typedef struct HIL_FORMAT_REQ_DATA_Ttag
{
  uint32_t  ulFlags;
  uint32_t  ulReserved;
} HIL_FORMAT_REQ_DATA_T;

typedef struct HIL_FORMAT_REQ_Ttag
{
  HIL_PACKET_HEADER                     tHead;                  /* packet header */
  HIL_FORMAT_REQ_DATA_T                 tData;
} HIL_FORMAT_REQ_T;
```

Format confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	8 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001ED7	HIL_FORMAT_CNF
Data			
ulExtended ErrorInfo	uint32_t	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_t		Offset the error was encountered on

Table 86: HIL_FORMAT_CNF_T – Format confirmation

Packet structure reference

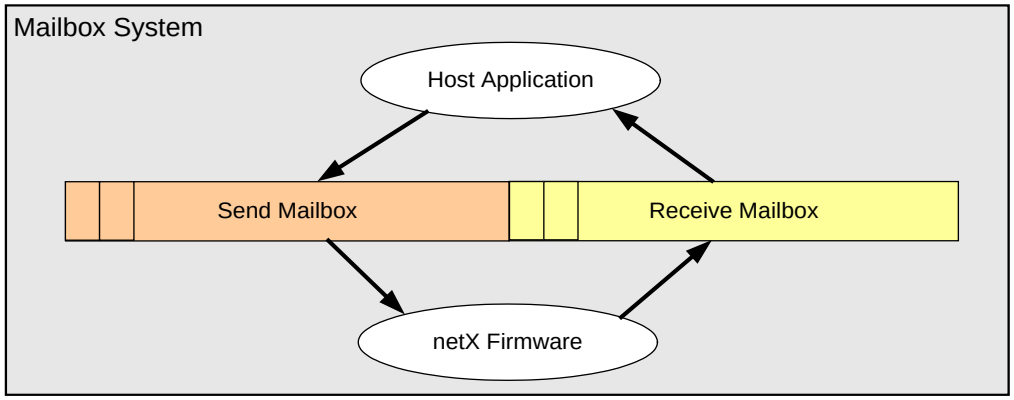
```
/* FORMAT CONFIRMATION */
#define HIL_FORMAT_CNF                      HIL_FORMAT_REQ+1

typedef struct HIL_FORMAT_CNF_DATA_Ttag
{
  /* Valid if format has failed during a full format with an error during
     erase / verify (ulSta = TLR_E_HIL_FORMAT_ERASE_FAILED or
     TLR_E_HIL_FORMAT_VERIFY_FAILED */
  uint32_t  ulExtendedErrorInfo;
  uint32_t  ulErrorOffset;
} HIL_FORMAT_CNF_DATA_T;

typedef struct HIL_FORMAT_CNF_Ttag
{
  HIL_PACKET_HEADER                     tHead;                  /* packet header */
  HIL_FORMAT_CNF_DATA_T                 tData;
} HIL_FORMAT_CNF_T;
```

3.17 General packet fragmentation

Two mailboxes are used to transfer a packet from the host application to the netX firmware or visa versa. Each mailbox has a limited size (= mailbox size). If the packet to be transferred is larger than the mailbox size, the packet has to be fragmented.

The mechanism of transferring packets in a fragmented manner is used

- in case the packet (size of packet header and user data) exceeds the size of the mailbox or
- in case the confirmation to a command packet has a variable data size, which exceeds the size of the mailbox.



Note: *Packet Fragmentation* is not a default mechanism for all packet commands. The general handling is described in this section and if supported it is explicitly noted in the packet command definition!

Handling for general packet fragmentation

Packet fragmentation is handled by the *ulExt* and *ulId* variable in the packet header. The *ulExt* variable defines whether a packet belongs to a sequence and indicates the state of a sequenced transfer (first/middle/last). *ulId* is used as a packet index within a sequence to ensure a strict packet order handling during a transfer.

Note: Fragmented packets must be sent in a strict order given by *ulId*. Out of order transfers are not supported.

Header variable	Description
ulExt	Indication of a sequenced packet transfer 0x00000000 = HIL_PACKET_SEQ_NONE — None sequenced (default) 0x00000080 = HIL_PACKET_SEQ_FIRST — First packet of a sequence 0x000000C0 = HIL_PACKET_SEQ_MIDDLE — Packet inside a sequence 0x00000040 = HIL_PACKET_SEQ_LAST — Last packet of a sequence
ulId	Packet number within a sequence ▪ Start value ▪ Incremented by one for each packet in a sequence

Table 87: Packet Fragmentation: Extension and Identifier Field

Example

Fragmented Transfer Host → netX Firmware (Initiated by host application)

Host application knows that data does not fit into the send mailbox.

Step	Direction (App | Task)	Description	*ulCmd*	*ulId*	ulExt
1	→	Command	CMD	X+0	HIL_PACKET_SEQ_FIRST
	←	Answer	CMD + 1		HIL_PACKET_SEQ_FIRST
2	→	Command	CMD	X+1	HIL_PACKET_SEQ_MIDDLE
	←	Answer	CMD + 1		HIL_PACKET_SEQ_MIDDLE
3	→	Command	CMD	X+2	HIL_PACKET_SEQ_MIDDLE
	←	Answer	CMD + 1		HIL_PACKET_SEQ_MIDDLE
…	…	…	…	…	…
n	→	Command	CMD	X+n	HIL_PACKET_SEQ_LAST
	←	Answer	CMD + 1		HIL_PACKET_SEQ_LAST

Table 88: Packet Fragmentation: Example - Host to netX Firmware

Fragmented Transfer netX Firmware → Host (Initiated by host application)

Host application does not know how many packets will be received.

Step	Direction (App | Task)	Description	*ulCmd*	*ulId*	ulExt
1	→	Command	CMD	X+0	HIL_PACKET_SEQ_NONE
	←	Answer	CMD + 1		**HIL_PACKET_SEQ_FIRST**
2	→	Command	CMD	X+1	HIL_PACKET_SEQ_MIDDLE
	←	Answer	CMD + 1		HIL_PACKET_SEQ_MIDDLE
3	→	Command	CMD	X+2	HIL_PACKET_SEQ_MIDDLE
	←	Answer	CMD + 1		HIL_PACKET_SEQ_MIDDLE
…	…	…	…	…	…
n	→	Command	CMD	X+n	HIL_PACKET_SEQ_MIDDLE
	←	Answer	CMD + 1		HIL_PACKET_SEQ_LAST

Table 89: Packet Fragmentation: Example - netX Firmware to Host

General Abort Handling

If an error is detected during a fragmented packet transfer, the transfer has to be aborted before the last packet is transferred. Examples for a fragmented packet transfers are file download and file upload functions.

Possible Errors:

- *ulId*, index skipped, used twice, or out of order
- *ulExt*, state out of order
- *ulSta* in the answer not zero, returned by the answering process

Note: If a service needs an abort handling will be mentioned in this manual.

Abort command

Variable	Type	Value / Range	Description
ulDest	uint32_t	n	Destination Address / Handle
ulSrc	uint32_t	n	Source Address / Handle
ulDestId	uint32_t	n	Destination Identifier
ulSrcId	uint32_t	n	Source Identifier
ulLen	uint32_t	**0**	Packet Data Length (in Byte)
ulId	uint32_t	**ANY**	Packet Identifier
ulSta	uint32_t	0	Packet State / Error
ulCmd	uint32_t	**CMD**	Packet Command / Confirmation
ulExt	uint32_t	**0x00000040**	Packet Extension Last Packet of Sequence
ulRout	uint32_t	0x00000000	Reserved (routing information)

Table 90: Packet Fragmentation: Abort command

Abort confirmation

Variable	Type	Value / Range	Description
ulDest	uint32_t	From Request	Destination Address / Handle
ulSrc	uint32_t	From Request	Source Address / Handle
ulDestId	uint32_t	From Request	Destination Identifier
ulSrcId	uint32_t	From Request	Source Identifier
ulLen	uint32_t	**0**	Packet Data Length (in Byte)
ulId	uint32_t	From Request	Packet Identifier
ulSta	uint32_t	**0**	Packet State / Error HIL_S_OK (always)
ulCmd	uint32_t	**CMD+1**	Packet Command / Confirmation
ulExt	uint32_t	**0x00000040**	Packet Extension Last Packet of Sequence
ulRout	uint32_t		Reserved (routing information)

Table 91: Packet Fragmentation: Abort confirmation

4 Communication Channel services

The following functions corresponding to information and functionalities of a **communication channel**.

4.1 Function overview

Communication Channel services		
Service	**Command definition**	**Page**
Communication Channel Information Blocks		
Read the Common Control Block of a channel	HIL_CONTROL_BLOCK_REQ	94
Read the Common Status Block of a channel	HIL_DPM_GET_COMMON_STATE_REQ	96
Read the Extended Status Block of a channel	HIL_DPM_GET_EXTENDED_STATE_REQ	98
Read Communication Flag States		
Read the communication flags of a specified communication channel	HIL_DPM_GET_COMFLAG_INFO_REQ	100
Read the I/O Process Data Image Size		
Read the configured size of the I/O process data image	HIL_GET_DPM_IO_INFO_REQ	102
Channel Initialization		
Re-initialize / re-configure a protocol stack	HIL_CHANNEL_INIT_REQ	105
Delete Protocol Stack Configuration		
Delete a actual configuration of a protocol stack	HIL_DELETE_CONFIG_REQ	107
Lock / Unlock Configuration		
Lock or unlock a configuration against changes	HIL_LOCK_UNLOCK_CONFIG_REQ	109
Start / Stop Communication		
Start or stop network communication	HIL_START_STOP_COMM_REQ	110
Channel Watchdog Time		
Read the actual watchdog time of a communication channel	HIL_GET_WATCHDOG_TIME_REQ	111
Set the watchdog time of a communication channel	HIL_SET_WATCHDOG_TIME_REQ	112

Table 92: Communication Channel services (function overview)

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 using packet fragmentation as described in section *General packet fragmentation* on page 90.

4.2.1 Read Common Control Block

Note: For a detailed description about the *Common Control Block*, see reference [1].

Read Common Control Block request

This packet is used to request the *Common Control Block*. The firmware returns the *Common Control Block* of the used Communication Channel and ignores the communication channel identifier `ulChannelId`. If the System Channel is used, the firmware will return the *Common Control Block* addressed by `ulChannelId`.

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	uint32_t	4	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00001E3A	HIL_CONTROL_BLOCK_REQ
Data			
`ulChannelId`	uint32_t	0 … 3	Communication Channel Number

Table 93: HIL_READ_COMM_CNTRL_BLOCK_REQ_T – Read Common Control Block request

Packet structure reference

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

typedef struct HIL_READ_COMM_CNTRL_BLOCK_REQ_DATA_Ttag
{
  uint32_t ulChannelId;                        /* channel identifier      */
} HIL_READ_COMM_CNTRL_BLOCK_REQ_DATA_T;

typedef struct HIL_READ_COMM_CNTRL_BLOCK_REQ_Ttag
{
  HIL_PACKET_HEADER                     tHead;     /* packet header        */
  HIL_READ_COMM_CNTRL_BLOCK_REQ_DATA_T  tData;     /* packet data          */
} HIL_READ_COMM_CNTRL_BLOCK_REQ_T;
```

Read Common Control Block confirmation

The following packet is returned by the firmware.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 8 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001E3B	HIL_CONTROL_BLOCK_CNF
Data			
tControl	Structure		Communication Control Block

Table 94: HIL_READ_COMM_CNTRL_BLOCK_CNF_T – Read Common Control Block confirmation

Packet structure reference

```
/* READ COMMUNICATION CONTROL BLOCK CONFIRMATION */
#define HIL_CONTROL_BLOCK_CNF                HIL_CONTROL_BLOCK_REQ+1

typedef struct HIL_READ_COMM_CNTRL_BLOCK_CNF_DATA_Ttag
{
  NETX_CONTROL_BLOCK                    tControl;  /* control block       */
} HIL_READ_COMM_CNTRL_BLOCK_CNF_DATA_T;

typedef struct HIL_READ_COMM_CNTRL_BLOCK_CNF_Ttag
{
  HIL_PACKET_HEADER                     tHead;     /* packet header       */
  HIL_READ_COMM_CNTRL_BLOCK_CNF_DATA_T  tData;     /* packet data         */
} HIL_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: For a detailed description about the *Common Status Block*, see reference [1].

Read Common Status Block request

This packet is used to request the *Common Status Block*. The firmware returns the *Common Status Block* of the used Communication Channel and ignores the communication channel identifier `ulChannelId`. If the System Channel is used, the firmware return the *Common Status Block* addressed by `ulChannelId`.

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	uint32_t	4	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00001EFC	HIL_DPM_GET_COMMON_STATE_REQ
Data			
`ulChannelId`	uint32_t	0 … 3	Communication Channel Number

Table 95: HIL_READ_COMMON_STS_BLOCK_REQ_T – Read Common Status Block request

Packet structure reference

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

typedef struct HIL_READ_COMMON_STS_BLOCK_REQ_DATA_Ttag
{
  uint32_t ulChannelId;                       /* channel identifier       */
} HIL_READ_COMMON_STS_BLOCK_REQ_DATA_T;

typedef struct HIL_READ_COMMON_STS_BLOCK_REQ_Ttag
{
  HIL_PACKET_HEADER                     tHead;     /* packet header        */
  HIL_READ_COMMON_STS_BLOCK_REQ_DATA_T  tData;     /* packet data          */
} HIL_READ_COMMON_STS_BLOCK_REQ_T;
```

Read Common Status Block confirmation

The following packet is returned by the firmware.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 64 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EFD	HIL_DPM_GET_COMMON_STATE_CNF
Data			
tCommonStatus	Structure		Common Status Block

Table 96: HIL_READ_COMMON_STS_BLOCK_CNF_T – Read Common Status Block confirmation

Packet structure reference

```
/* READ COMMON STATUS BLOCK CONFIRMATION */
#define HIL_DPM_GET_COMMON_STATE_CNF         HIL_DPM_GET_COMMON_STATE_REQ+1

typedef struct HIL_READ_COMMON_STS_BLOCK_CNF_DATA_Ttag
{
  NETX_COMMON_STATUS_BLOCK              tCommonStatus;  /* common status   */
} HIL_READ_COMMON_STS_BLOCK_CNF_DATA_T;

typedef struct HIL_READ_COMMON_STS_BLOCK_CNF_Ttag
{
  HIL_PACKET_HEADER                     tHead;  /* packet header          */
  HIL_READ_COMMON_STS_BLOCK_CNF_DATA_T  tData;  /* packet data            */
} HIL_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: For a detailed description about the *Extended Status Block*, see reference [1].

This packet is used to request the *Extended Status Block*. The firmware returns the *Extended Status Block* of the used Communication Channel and ignores the communication channel identifier `ulChannelId`. If the System Channel is used, the firmware returns the *Extended Status Block* addressed by `ulChannelId`.

This service does not support fragmentation.

Read Extended Status Block request

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	uint32_t	12	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00001EFE	HIL_DPM_GET_EXTENDED_STATE_REQ
Data			
`ulOffset`	uint32_t	0 … 431	Byte offset in extended status block structure
`ulDataLen`	uint32_t	1 … 432	Length in byte read
`ulChannel Index`	uint32_t	0 … 3	Communication Channel Number

Table 97: HIL_DPM_GET_EXTENDED_STATE_REQ_T – Read Extended Status Block request

Packet structure reference

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

typedef struct HIL_DPM_GET_EXTENDED_STATE_REQ_DATA_Ttag
{
  uint32_t ulOffset;           /* offset in extended status block        */
  uint32_t ulDataLen;          /* size of block to read                  */
  uint32_t ulChannelIndex;     /* channel number                         */
} HIL_DPM_GET_EXTENDED_STATE_REQ_DATA_T;

typedef struct HIL_DPM_GET_EXTENDED_STATE_REQ_Ttag
{
  HIL_PACKET_HEADER                      tHead;   /* packet header       */
  HIL_DPM_GET_EXTENDED_STATE_REQ_DATA_T  tData;   /* packet data         */
} HIL_DPM_GET_EXTENDED_STATE_REQ_T;
```

Read Extended Status Block confirmation

The following packet is returned by the firmware.

Variable	Type	Value / Range	Description
ulLen	uint32_t	 1 … 432 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EFF	HIL_DPM_GET_EXTENDED_STATE_CNF
Data			
ulOffset	uint32_t	0 … 431	Byte offset in extended status block structure
ulDataLen	uint32_t	1 … 432	Length in byte
abData[432]	uint8_t	0 … n	Extended Status Block data

Table 98: HIL_DPM_GET_EXTENDED_STATE_CNF_T – Read Extended Status Block confirmation

Packet structure reference

```
/* READ EXTENDED STATUS BLOCK CONFIRMATION */
#define HIL_DPM_GET_EXTENDED_STATE_CNF       HIL_DPM_GET_EXTENDED_STATE_REQ+1

typedef struct HIL_DPM_GET_EXTENDED_STATE_CNF_DATA_Ttag
{
  uint32_t ulOffset;           /* offset in extended status block        */
  uint32_t ulDataLen;          /* size of block returned                 */
  uint8_t  abData[432];        /* data block                             */
} HIL_DPM_GET_EXTENDED_STATE_CNF_DATA_T;

typedef struct HIL_DPM_GET_EXTENDED_STATE_CNF_Ttag
{
  HIL_PACKET_HEADER                      tHead;     /* packet header      */
  HIL_DPM_GET_EXTENDED_STATE_CNF_DATA_T  tData;     /* packet data        */
} HIL_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

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000000	HIL_PACKET_DEST_SYSTEM
`ulLen`	uint32_t	4	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00001EFA	HIL_DPM_GET_COMFLAG_INFO_REQ
Data			
`ulAreaIndex`	uint32_t	0 … 5	Area Index (see below)

Table 99: HIL_DPM_GET_COMFLAG_INFO_REQ_T – DPM Get ComFlag Info request

Packet structure reference

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

typedef struct HIL_DPM_GET_COMFLAG_INFO_REQ_DATA_Ttag
{
  uint32_t                          ulAreaIndex;           /* area index */
} HIL_DPM_GET_COMFLAG_INFO_REQ_DATA_T;

typedef struct HIL_DPM_GET_COMFLAG_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER_T                   tHead;             /* packet header */
  HIL_DPM_GET_COMFLAG_INFO_REQ_DATA_T   tData;             /* packet data */
} HIL_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.

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

Table 100: Area Index

DPM Get ComFlag Info confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	 12 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00001EFB	HIL_DPM_GET_COMFLAG_INFO_CNF
Data			
ulAreaIndex	uint32_t	0 … 5	Area Index (see above)
ulNetxComFlag	uint32_t	Bit Field	Current netX Communication Flags
ulHostComFlag	uint32_t	Bit Field	Current Host Communication Flags

Table 101: HIL_DPM_GET_COMFLAG_INFO_CNF_T – DPM Get ComFlag Info confirmation

Packet structure reference

```
/* DPM GET COMFLAG INFO CONFIRMATION */
#define HIL_DPM_GET_COMFLAG_INFO_CNF       HIL_DPM_GET_COMFLAG_INFO_REQ+1

typedef struct HIL_DPM_GET_COMFLAG_INFO_CNF_DATA_Ttag
{
  uint32_t ulAreaIndex;                         /* area index */
  uint32_t ulNetxComFlag;
  uint32_t ulHostComFlag;
} HIL_DPM_GET_COMFLAG_INFO_CNF_DATA_T;

typedef struct HIL_DPM_GET_COMFLAG_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER_T                 tHead;    /* packet header */
  HIL_DPM_GET_COMFLAG_INFO_CNF_DATA_T tData;    /* packet data */
} HIL_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 that are defined by the fieldbus protocol configuration.

Note: Some of the protocol stacks are able to map additional state information into the I/O data image. The **additional** length must be obtained from the extended state block information (see section *Read Extended Status Block* on page 98) because this service does not report the additional length.

Note: If the process data is configured to be input only or output only, the confirmation packet will report two blocks (input and output) stating that the unused block has a length of 0.

Get DPM I/O Information request

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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	uint32_t	0x00002F0C	HIL_GET_DPM_IO_INFO_REQ

Table 102: HIL_GET_DPM_IO_INFO_REQ_T – Get DPM I/O Information request

Packet structure reference

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

typedef struct HIL_GET_DPM_IO_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header     */
} HIL_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!

Variable	Type	Value / Range	Description
ulLen	uint32_t	4+(20 * n) 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulId	uint32_t	From Request	Packet Identification as Unique Number
ulSta	uint32_t	See Below	Status / Error Code see Section 6
ulCmd	uint32_t	0x00002F0D	HIL_GET_DPM_IO_INFO_CNF
ulExt	uint32_t	0x00000000 0x00000080 0x000000C0 0x00000040	Extension No Sequenced Packet First Packet of Sequence Sequenced Packet Last Packet of Sequence
Data			
ulNumIOBlock Info	uint32_t	0 … 10	Number **n** of Block Definitions Below
atIoBlockInfo [2]	Array of Structure		I/O Block Definition Structure(s) HIL_DPM_IO_BLOCK_INFO

Table 103: HIL_GET_DPM_IO_INFO_CNF_T – Get DPM I/O Information confirmation

Packet structure reference

```
/* GET DPM I/O INFORMATION CONFIRMATION */
#define HIL_GET_DPM_IO_INFO_CNF             HIL_GET_DPM_IO_INFO_REQ+1

typedef struct HIL_DPM_IO_BLOCK_INFO_Ttag
{
  uint32_t ulSubblockIndex;  /* index of sub block                        */
  uint32_t ulType;           /* type of sub block                         */
  uint16_t usFlags;          /* flags of the sub block                    */
  uint16_t usReserved;       /* reserved                                  */
  uint32_t ulOffset;         /* offset                                    */
  uint32_t ulLength;         /* length of I/O data in bytes               */
} HIL_DPM_IO_BLOCK_INFO_T;

typedef struct HIL_GET_DPM_IO_INFO_CNF_DATA_Ttag
{
  uint32_t                 ulNumIOBlockInfo; /* Number of IO Block Info        */
  HIL_DPM_IO_BLOCK_INFO_T  atIoBlock[2];     /* Array of I/O Block information */
} HIL_GET_DPM_IO_INFO_CNF_DATA_T;

typedef struct HIL_GET_DPM_IO_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header             */
  HIL_GET_DPM_IO_INFO_CNF_DATA_T   tData;   /* packet data               */
} HIL_GET_DPM_IO_INFO_CNF_T;
```

Variable	Type	Value / Range	Description
ulSubblockIndex	uint32_t	5, 6	Index of sub block The value identifies the index of the sub block. This field is only informative and shall not be used by an application. Value 5 for standard output image. Value 6 for standard input image.

ulType	uint32_t		Type of sub block HIL_BLOCK_* type definitions, see Hil_DualPortMemory.h.
usFlags	uint16_t		Flags of the sub block HIL_DIRECTION_* and HIL_TRANSMISSION_TYPE_* type definitions, see Hil_DualPortMemory.h
usReserved	uint16_t		Reserved
ulOffset	uint32_t	0	Offset Offset is always 0, even if the application has not configured any I/O data to offset 0. However, for legacy reasons the field may not be 0 in some cases.
ulLength	uint32_t		Length of I/O data in bytes Highest offset address of input data or output data used in the process data image (starting with offset 0, even if the application has not configured any I/O data to offset 0).

Table 104: Structure HIL_DPM_IO_BLOCK_INFO

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.

This service may be negatively responded by a protocol stack to indicate that no configuration was applied e.g. because no configuration is available that can be used or because no valid MAC address is available.

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

In order to avoid race conditions in firmware (e.g. mailbox events generated by firmware are not recognized by the application), best practice is to use the following flow diagram to perform a ChannelInit.

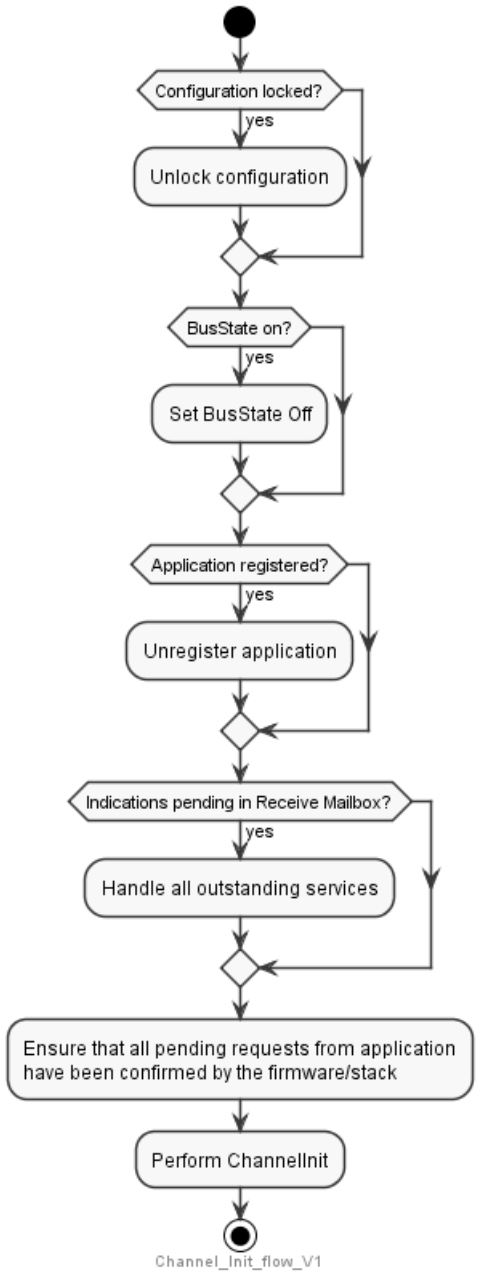


Figure 3: Flow chart ChannelInit (Best practise pattern for the host application)

Channel Initialization request

The packet is send through the channel mailbox.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	uint32_t	0x00002F80	HIL_CHANNEL_INIT_REQ

Table 105: HIL_CHANNEL_INIT_REQ_T – Channel Initialization request

Packet structure reference

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

typedef struct HIL_CHANNEL_INIT_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header             */
} HIL_CHANNEL_INIT_REQ_T;
```

Channel Initialization confirmation

The channel firmware returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F81	HIL_CHANNEL_INIT_CNF

Table 106: HIL_CHANNEL_INIT_CNF_T – Channel Initialization confirmation

Packet structure reference

```
/* CHANNEL INITIALIZATION CONFIRMATION */
#define HIL_CHANNEL_INIT_CNF                HIL_CHANNEL_INIT_REQ+1

typedef struct HIL_CHANNEL_INIT_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header             */
} HIL_CHANNEL_INIT_CNF_T;
```

4.6 Delete Protocol Stack Configuration

A protocol stack can be configured

- via packet services (Set Configuration packets) or
- via a configuration database file.

The application can use this packet to delete the configuration in the RAM.

This service will overwrite remanent data stored in non-volatile memory with default data. However, some legacy implementations may not do so.

Consequence for	Configured via packets	Configured via configuration database
Configuration	The configuration stored in RAM will be deleted. The application has to use the Set Configuration service again. Otherwise (after a channel initialization) the protocol stack won't startup properly due to the missing configuration.	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. For details, see section *Delete a File* page 52.
Remanent data	The remanent data will be reset to default values.	The remanent data will be reset to default values.

Table 107: Delete protocol stack configuration

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

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.

Variable	Type	Value / Range	Description
`ulDest`	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
`ulCmd`	uint32_t	0x00002F14	HIL_DELETE_CONFIG_REQ

Table 108: HIL_DELETE_CONFIG_REQ_T – Delete Configuration request

Packet structure reference

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

typedef struct HIL_DELETE_CONFIG_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_DELETE_CONFIG_REQ_T;
```

Delete Configuration confirmation

The system returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F15	HIL_DELETE_CONFIG_CNF

Table 109: HIL_DELETE_CONFIG_CNF_T – Delete Configuration confirmation

Packet structure reference

```
/* DELETE CONFIGURATION CONFIRMATION */
#define HIL_DELETE_CONFIG_CNF                HIL_DELETE_CONFIG_REQ+1

typedef struct HIL_DELETE_CONFIG_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header             */
} HIL_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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F32	HIL_LOCK_UNLOCK_CONFIG_REQ
Data			
ulParam	uint32_t	0x00000001 0x00000002	Parameter Lock Configuration Unlock Configuration

Table 110: HIL_LOCK_UNLOCK_CONFIG_REQ_T – Lock / Unlock Config request

Packet structure reference

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

typedef struct HIL_LOCK_UNLOCK_CONFIG_REQ_DATA_Ttag
{
  uint32_t ulParam;                            /* lock/unlock parameter  */
} HIL_LOCK_UNLOCK_CONFIG_REQ_DATA_T;

typedef struct HIL_LOCK_UNLOCK_CONFIG_REQ_Ttag
{
  HIL_PACKET_HEADER                   tHead;   /* packet header          */
  HIL_LOCK_UNLOCK_CONFIG_REQ_DATA_T   tData;   /* packet data            */
} HIL_LOCK_UNLOCK_CONFIG_REQ_T;
```

Lock / Unlock Config confirmation

The channel firmware returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F33	HIL_LOCK_UNLOCK_CONFIG_CNF

Table 111: HIL_LOCK_UNLOCK_CONFIG_CNF_T – Lock / Unlock Config confirmation

Packet structure reference

```
/* LOCK – UNLOCK CONFIGURATION CONFIRMATION */
#define HIL_LOCK_UNLOCK_CONFIG_CNF          HIL_LOCK_UNLOCK_CONFIG_REQ+1

typedef struct HIL_LOCK_UNLOCK_CONFIG_CNF_Ttag
{
  HIL_PACKET_HEADER    tHead;            /* packet header              */
} HIL_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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F30	HIL_START_STOP_COMM_REQ
Data			
ulParam	uint32_t	0x00000001 0x00000002	Parameter HIL_START_STOP_COMM_PARAM_START HIL_START_STOP_COMM_PARAM_STOP

Table 112: HIL_START_STOP_COMM_REQ_T – Start / Stop Communication request

Packet structure reference

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

#define HIL_START_STOP_COMM_PARAM_START 0x00000001
#define HIL_START_STOP_COMM_PARAM_STOP  0x00000002

typedef struct HIL_START_STOP_COMM_REQ_DATA_Ttag
{
  uint32_t ulParam;                      /* start/stop communication      */
} HIL_START_STOP_COMM_REQ_DATA_T;

typedef struct HIL_START_STOP_COMM_REQ_Ttag
{
  HIL_PACKET_HEADER               tHead;       /* packet header          */
  HIL_START_STOP_COMM_REQ_DATA_T  tData;       /* packet data            */
} HIL_START_STOP_COMM_REQ_T;
```

Start / Stop Communication confirmation

The firmware returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F31	HIL_START_STOP_COMM_CNF

Table 113: HIL_START_STOP_COMM_CNF_T – Start / Stop Communication confirmation

Packet structure reference

```
/* START – STOP COMMUNICATION CONFIRMATION    */
#define HIL_START_STOP_COMM_CNF              HIL_START_STOP_COMM_REQ+1

typedef struct HIL_START_STOP_COMM_CNF_Ttag
{
  HIL_PACKET_HEADER     tHead;            /* packet header               */
} HIL_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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	uint32_t	0x00002F02	HIL_GET_WATCHDOG_TIME_REQ

Table 114: HIL_GET_WATCHDOG_TIME_REQ_T – Get Watchdog Time request

Packet structure reference

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

typedef struct HIL_GET_WATCHDOG_TIME_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header     */
} HIL_GET_WATCHDOG_TIME_REQ_T;
```

Get Watchdog Time confirmation

The system channel returns the following packet.

Variable	Type	Value / Range	Description
ulLen	uint32_t	4 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F03	HIL_GET_WATCHDOG_TIME_CNF
Data			
ulWdgTime	uint32_t	0 20 … 0xFFFF	Watchdog Time in milliseconds [ms] = not set 20 > WDT < 0xFFFF

Table 115: HIL_GET_WATCHDOG_TIME_CNF_T – Get Watchdog Time confirmation

Packet structure reference

```
/* GET WATCHDOG TIME CONFIRMATION */
#define HIL_GET_WATCHDOG_TIME_CNF            HIL_GET_WATCHDOG_TIME_REQ+1

typedef struct HIL_GET_WATCHDOG_TIME_CNF_DATA_Ttag
{
  uint32_t             ulWdgTime;   /* current watchdog time            */
} HIL_GET_WATCHDOG_TIME_CNF_DATA_T;

typedef struct HIL_GET_WATCHDOG_TIME_CNF_Ttag
{
  HIL_PACKET_HEADER                  tHead;     /* packet header          */
  HIL_GET_WATCHDOG_TIME_CNF_DATA_T   tData;     /* packet data            */
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F04	HIL_SET_WATCHDOG_TIME_REQ
Data			
ulWdgTime	uint32_t	0 20 … 65535	Watchdog Time Watchdog inactive Watchdog time in milliseconds

Table 116: HIL_SET_WATCHDOG_TIME_REQ_T – Set Watchdog Time request

Packet structure reference

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

typedef struct HIL_SET_WATCHDOG_TIME_REQ_DATA_Ttag
{
  /** watchdog time in milliseconds */
  uint32_t ulWdgTime;
} HIL_SET_WATCHDOG_TIME_REQ_DATA_T;

typedef struct HIL_SET_WATCHDOG_TIME_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;
  HIL_SET_WATCHDOG_TIME_REQ_DATA_T  tData;
} HIL_SET_WATCHDOG_TIME_REQ_T;
```

Set Watchdog Time confirmation

The system channel returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F05	HIL_SET_WATCHDOG_TIME_CNF

Table 117: HIL_SET_WATCHDOG_TIME_CNF_T – Set Watchdog Time confirmation

Packet structure reference

```
/* SET WATCHDOG TIME CONFIRMATION */
#define HIL_SET_WATCHDOG_TIME_CNF    HIL_SET_WATCHDOG_TIME_REQ+1

typedef struct HIL_SET_WATCHDOG_TIME_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_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	HIL_SET_HANDSHAKE_CONFIG_REQ	114
Modify Configuration Settings		
Set protocol stack configuration parameters to new values	HIL_SET_FW_PARAMETER_REQ	119
Network Connection State		
Obtain a list of slave which are configured, active or faulted	HIL_GET_SLAVE_HANDLE_REQ	123
Obtain a slave connection information	HIL_GET_SLAVE_CONN_INFO_REQ	125
Protocol Stack Notifications / Indications		
Register an application to be able to receive notifications from a protocol stack	HIL_REGISTER_APP_REQ	128
Unregister an application from receiving notifications	HIL_UNREGISTER_APP_REQ	129
Link Status Changed Service		
Activate a link status change notification	HIL_LINK_STATUS_CHANGE_IND	130
Perform a Bus Scan		
Scan for available devices on the fieldbus devices	HIL_BUSSCAN_REQ	132
Get Information about a Fieldbus Device		
Read the fieldbus depending information of a device	HIL_GET_DEVICE_INFO_REQ	134
Configuration in Run		
Verify a modified configuration database file	HIL_VERIFY_DATABASE_REQ	136
Activate the modified configuration	HIL_ACTIVATE_DATABASE_REQ	138

Table 118: Protocol stack services (function overview)

5.2 DPM Handshake Configuration

The host application has the option to modify the netX firmware specific behavior of the IO handshake and the Sync handshake. Depending on the protocol stack, this service is or is not implemented.

For a description of the handshake modes and the handling in general, see reference [1].

Note: To protect the netX CPU from unexpected overload scenarios, a firmware may have implemented a protection mechanism. This mechanism will only allow a specific amount of IO data exchanges per time. If too many IO exchange requests are detected by the firmware, the firmware will not handle a request directly but instead wait for a specific amount of time until the request will be handled.

5.2.1 Set Handshake Configuration

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

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 exchange with the protocol stack could lead into unpredictable states in the I/O synchronisation.

Set Handshake Configuration request

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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	20	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F34	HIL_SET_HANDSHAKE_CONFIG_REQ
Data			
bPDInHskMode	uint8_t		Input process data handshake mode
bPDInSource	uint8_t	0	Input process data trigger source; unused, set to zero
usPDInErrorTh	uint16_t		Threshold for input process data handshake handling errors
bPDOutHskMode	uint8_t		Output process data handshake mode
bPDOutSource	uint8_t	0	Output process data trigger source; unused, set to zero
usPDOutErrorTh	uint16_t	0 … 0xFFFF	Threshold for output process data handshake handling errors
bSyncHskMode	uint8_t		Synchronization handshake mode
bSyncSource	uint8_t		Synchronization source
usSyncErrorTh	uint16_t	0 … 0xFFFF	Threshold for synchronization handshake handling errors
aulReserved[2]	uint32_t	0	Reserved for future use; set to zero

Table 119: HIL_SET_HANDSHAKE_CONFIG_REQ_T - Set Handshake Configuration request

Packet structure reference

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

typedef struct HIL_SET_HANDSHAKE_CONFIG_REQ_DATA_Ttag
{
  uint8_t  bPDInHskMode;   /* input process data handshake mode           */
  uint8_t  bPDInSource;    /* input process data trigger source           */
  uint16_t usPDInErrorTh;  /* threshold for input data handshake handling errors     */
  uint8_t  bPDOutHskMode;  /* output process data handshake mode          */
  uint8_t  bPDOutSource;   /* output process data trigger source          */
  uint16_t usPDOutErrorTh; /* threshold for output data handshake handling errors    */
  uint8_t  bSyncHskMode;   /* synchronization handshake mode              */
  uint8_t  bSyncSource;    /* synchronization source                      */
  uint16_t usSyncErrorTh;  /* threshold for sync handshake handling errors*/
  uint32_t aulReserved[2]; /* reserved for future use                     */
} HIL_SET_HANDSHAKE_CONFIG_REQ_DATA_T;

typedef struct HIL_SET_HANDSHAKE_CONFIG_REQ_Ttag
{
  HIL_PACKET_HEADER                       tHead;   /* packet header      */
  HIL_SET_HANDSHAKE_CONFIG_REQ_DATA_T     tData;   /* packet data        */
} HIL_SET_HANDSHAKE_CONFIG_REQ_T;
```

Set Handshake Configuration confirmation

The firmware returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F35	HIL_SET_HANDSHAKE_CONFIG_CNF

Table 120: HIL_SET_HANDSHAKE_CONFIG_CNF_T - Set Handshake Configuration confirmation

Packet structure reference

```
/* SET HANDSHAKE CONFIGURATION CONFIRMATION */
#define HIL_SET_HANDSHAKE_CONFIG_CNF         HIL_SET_HANDSHAKE_CONFIG_REQ+1

typedef struct HIL_SET_HANDSHAKE_CONFIG_CNF_Ttag
{
  HIL_PACKET_HEADER                       tHead;   /* packet header      */
} HIL_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

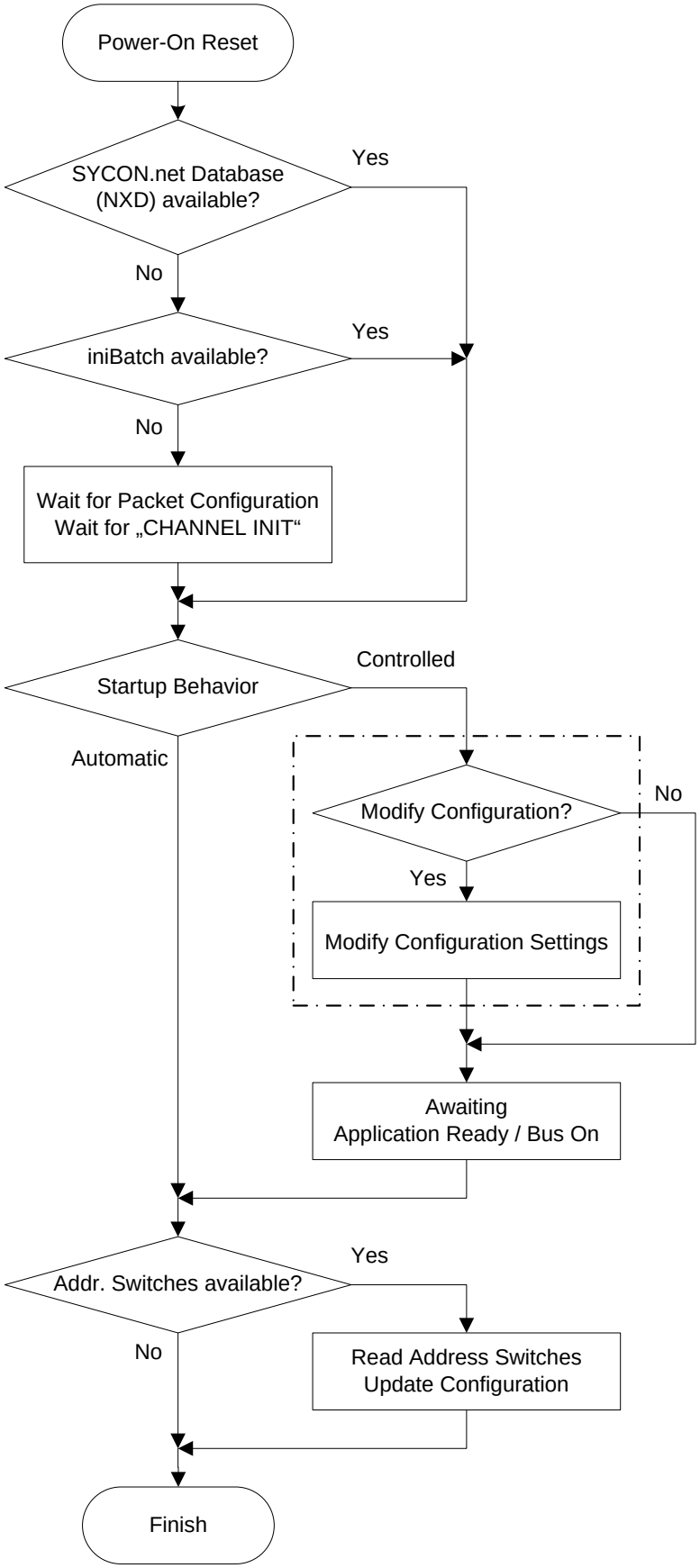


Figure 4: Flowchart Modify Configuration Settings

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 109).

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 the application 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_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
`ulLen`	uint32_t	8 + n	Packet Data Length (in Bytes)
`ulCmd`	uint32_t	0x00002F86	HIL_SET_FW_PARAMETER_REQ
Data			
`ulParameterID`	uint32_t	0 … 0xFFFFFFFF	Parameter identifier, see Table 122 and Table 123
`ulParameterLen`	uint32_t	n	Length of abParameter in byte
`abParameter[4]`	uint8_t	m	Parameter value, byte array

Table 121: HIL_SET_FW_PARAMETER_REQ_T – Set Parameter request

Packet structure reference

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

typedef struct HIL_SET_FW_PARAMETER_REQ_DATA_Ttag
{
  uint32_t ulParameterID;                  /* parameter identifier       */
  uint32_t ulParameterLen;                 /* parameter length           */
  uint8_t  abParameter[4];                 /* parameter                  */
} HIL_SET_FW_PARAMETER_REQ_DATA_Ttag;

typedef struct HIL_SET_FW_PARAMETER_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header              */
  HIL_SET_FW_PARAMETER_REQ_DATA_T  tData;   /* packet data                */
} HIL_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 122: Encoding Parameter Identifier

The following parameter identifiers are defined.

Name	Code	Type	Size	Description of Parameter
PID_STATION_ADDRESS	0x30000001	uint32_t	4 Byte	Station Address
PID_BAUDRATE	0x30000002	uint32_t	4 Byte	Baud Rate
PID_PN_NAME_OF_STATION	0x30015001	uint8_t	240 Byte	PROFINET: Name of Station
PID_ECS_DEVICE_IDENTIFICATION	0x30009001	uint16_t	4 Byte	EtherCAT: Value for Explicit Device Identification
PID_ECS_SCND_STATION_ADDRESS	0x30009002	uint16_t	4 Byte	EtherCAT: Second Station Address
All other codes are reserved for future use.				

Table 123: Defined Parameter Identifier

Set Parameter confirmation

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

Variable	Type	Value / Range	Description
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F87	HIL_SET_FW_PARAMETER_CNF

Table 124: HIL_SET_FW_PARAMETER_CNF_T – Set Parameter confirmation

Packet structure reference

```
/* SET FIRMWARE PARAMETER CONFIRMATION */
#define HIL_SET_FW_PARAMETER_CNF           HIL_SET_FW_PARAMETER_REQ+1

typedef struct HIL_SET_FW_PARAMETER_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_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* (*HIL_LIST_CONF_SLAVES*)
- *List of Activated Slaves* (*HIL_LIST_ACTV_SLAVES*)
- *List of Faulted Slaves* (*HIL_LIST_FAULTED_SLAVES*)

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F08	HIL_GET_SLAVE_HANDLE_REQ
Data			
ulParam	uint32_t	 0x00000001 0x00000002 0x00000003	Parameter HIL_LIST_CONF_SLAVES HIL_LIST_ACTV_SLAVES HIL_LIST_FAULTED_SLAVES

Table 125: HIL_PACKET_GET_SLAVE_HANDLE_REQ_T – Get Slave Handle request

Packet structure reference

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

/* LIST OF SLAVES */
#define HIL_LIST_CONF_SLAVES                 0x00000001
#define HIL_LIST_ACTV_SLAVES                 0x00000002
#define HIL_LIST_FAULTED_SLAVES              0x00000003

typedef struct HIL_PACKET_GET_SLAVE_HANDLE_REQ_DATA_Ttag
{
  uint32_t ulParam;                          /* type of list               */
} HIL_PACKET_GET_SLAVE_HANDLE_REQ_DATA_T;

typedef struct HIL_PACKET_GET_SLAVE_HANDLE_REQ_Ttag
{
  HIL_PACKET_HEADER                        tHead;     /* packet header     */
  HIL_PACKET_GET_SLAVE_HANDLE_REQ_DATA_T   tData;     /* packet data       */
} HIL_PACKET_GET_SLAVE_HANDLE_REQ_T;
```

Get Slave Handle confirmation

This is the answer to the *HIL_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*).

Variable	Type	Value / Range	Description
`ulLen`	uint32_t	4 * (1 + n) 0	Packet Data Length (in Bytes) If `ulSta` = `HIL_S_OK` Otherwise
`ulSta`	uint32_t	See Below	Status / Error Code, see Section 6
`ulCmd`	uint32_t	0x00002F09	HIL_GET_SLAVE_HANDLE_CNF
Data			
`ulParam`	uint32_t	0x00000001 0x00000002 0x00000003	Parameter HIL_LIST_CONF_SLAVES HIL_LIST_ACTV_SLAVES HIL_LIST_FAULTED_SLAVES
`aulHandle[1]`	uint32_t	0 … 0xFFFFFFFF	Slave Handle, Number of Handles is **n**

Table 126: HIL_PACKET_GET_SLAVE_HANDLE_CNF_T – Get Slave Handle confirmation

Packet structure reference

```
/* GET SLAVE HANDLE CONFIRMATION */
#define HIL_GET_SLAVE_HANDLE_CNF             HIL_GET_SLAVE_HANDLE_REQ+1

typedef struct HIL_PACKET_GET_SLAVE_HANDLE_CNF_DATA_Ttag
{
  uint32_t ulParam;                         /* type of list               */
  /* list of handles follows here                                         */
  uint32_t aulHandle[1];
} HIL_PACKET_GET_SLAVE_HANDLE_CNF_DATA_T

typedef struct HIL_PACKET_GET_SLAVE_HANDLE_CNF_Ttag
{
  HIL_PACKET_HEADER                       tHead;    /* packer header     */
  HIL_PACKET_GET_SLAVE_HANDLE_CNF_DATA_T  tData;    /* packet data       */
} HIL_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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F0A	HIL_GET_SLAVE_CONN_INFO_REQ
Data			
ulHandle	uint32_t	0 … 0xFFFFFFFF	Slave Handle

Table 127: HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_T – Get Slave Connection Information request

Packet structure reference

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

typedef struct HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_Ttag
{
  uint32_t ulHandle;                                    /* slave handle  */
} HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_T;

typedef struct HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER                             tHead;   /* packer header */
  HIL_PACKET_GET_SLAVE_CONN_INFO_REQ_DATA_T     tData;   /* packet data   */
} HIL_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.

Variable	Type	Value / Range	Description
ulLen	uint32_t	8+sizeof(*slave data*) 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F0B	HIL_GET_SLAVE_CONN_INFO_CNF
Data			
ulHandle	uint32_t	0 … 0xFFFFFFFF	Slave Handle
ulStructID	uint32_t	0 … 0xFFFFFFFF	Structure Identification Number
slave data	Structure	n	Fieldbus Specific Slave Status Information (Refer to Fieldbus Documentation)

Table 128: HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_T – Get Slave Connection Information conformation

Packet structure reference

```
/* GET SLAVE CONNECTION INFORMATION CONFIRMATION */
#define HIL_GET_SLAVE_CONN_INFO_CNF          HIL_GET_SLAVE_CONN_INFO_REQ+1

typedef struct HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_Ttag
{
  uint32_t ulHandle;                /* slave handle                      */
  uint32_t ulStructID;              /* structure identification number   */
  /* fieldbus specific slave status information follows here             */
} HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_T;

typedef struct HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER                             tHead;    /* packet header     */
  HIL_PACKET_GET_SLAVE_CONN_INFO_CNF_DATA_T     tData;    /* packet data  */
} HIL_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 notifications / indications are used to inform the application about state changes and other protocol stack relevant information.

This section describes the method on how to register / unregister an application to the protocol stack in order to activate and receive notifications / indications via the mailbox system.

Note: Available information (as notifications / indications) depends on the protocol stack. Please consult the corresponding Protocol API manual.

During the registration of the application, notifications will be automatically activated. From this point of time, the application **must process incoming notification / indication packets**. If an application does not process the notification / indication after registration, the protocol stack internal service will time-out which can 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 attempts are valid and `ulSrc` is copied into every notification / indication packet send to the host application to help identifying the intended receiver.

Ethernet-based protocol stacks will automatically issue the *Link Status Changed Service* (see page 130) after the application has registered.

Note: Only one application is able to register with the protocol stack at a time. Further register attempts in parallel will be rejected by the protocol stack.

5.5.1 Register Application

Register Application request

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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulSrc	uint32_t	n	Unique application identifier
ulCmd	uint32_t	0x00002F10	HIL_REGISTER_APP_REQ

Table 129: HIL_REGISTER_APP_REQ_T – Register Application request

Packet structure reference

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

typedef struct HIL_REGISTER_APP_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_REGISTER_APP_REQ_T;
```

Register Application confirmation

The system channel returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F11	HIL_REGISTER_APP_CNF

Table 130: HIL_REGISTER_APP_CNF_T – Register Application confirmation

Packet structure reference

```
/* REGISTER APPLICATION CONFIRMATION */
#define HIL_REGISTER_APP_CNF                HIL_REGISTER_APP_REQ+1

typedef struct HIL_REGISTER_APP_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_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.

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulSrc	uint32_t	n	used during registration
ulCmd	uint32_t	0x00002F12	HIL_UNREGISTER_APP_REQ

Table 131: HIL_UNREGISTER_APP_REQ_T – Unregister Application request

Packet structure reference

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

typedef struct HIL_UNREGISTER_APP_REQ_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_UNREGISTER_APP_REQ_T;
```

Unregister Application confirmation

The system channel returns the following packet.

Variable	Type	Value / Range	Description
ulSta	uint32_t	See Below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F13	HIL_UNREGISTER_APP_CNF

Table 132: HIL_UNREGISTER_APP_CNF_T – Unregister Application confirmation

Packet structure reference

```
/* UNREGISTER APPLICATION CONFIRMATION */
#define HIL_UNREGISTER_APP_CNF              HIL_UNREGISTER_APP_REQ+1

typedef struct HIL_UNREGISTER_APP_CNF_Ttag
{
  HIL_PACKET_HEADER                tHead;   /* packet header            */
} HIL_UNREGISTER_APP_CNF_T;
```

5.6 Link Status Changed Service

This service is used to inform an application 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*).

An Ethernet-based protocol stack will automatically generate this indication after the application has used the *Register Application* service (page 128).

This command depends on the used protocol stack, see corresponding Protocol API manual if this command is supported.

Link Status Change indication

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	32	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F8A	HIL_LINK_STATUS_CHANGE_IND
Data			
atLinkData[2]	Structure		Link Status Information

Structure Information: *HIL_LINK_STATUS_T*			
ulPort	uint32_t		Number of the port
fIsFullDuplex	uint32_t		Non-zero if full duplex is used
fIsLinkUp	uint32_t		Non-zero if link is up
ulSpeed	uint32_t	 0 10 100	Speed of the link No link 10MBit 100Mbit

Table 133: HIL_LINK_STATUS_CHANGE_IND_T – Link Status Change indication

Packet structure reference

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

typedef struct HIL_LINK_STATUS_Ttag
{
  uint32_t        ulPort;           /*!< Port the link status is for */
  uint32_t        fIsFullDuplex;    /*!< If a full duplex link is available on this port
*/
  uint32_t        fIsLinkUp;        /*!< If a link is available on this port */
  uint32_t        ulSpeed;          /*!< Speed of the link \n\n
                                     \valueRange
                                     0:   No link \n
                                     10:  10MBit  \n
                                     100: 100MBit \n */
} HIL_LINK_STATUS_T;

typedef struct HIL_LINK_STATUS_CHANGE_IND_DATA_Ttag
{
  HIL_LINK_STATUS_T  atLinkData[2];
} HIL_LINK_STATUS_CHANGE_IND_DATA_T;

typedef struct HIL_LINK_STATUS_CHANGE_IND_Ttag
{
  HIL_PACKET_HEADER                tHead;
  HIL_LINK_STATUS_CHANGE_IND_DATA_T tData;
} HIL_LINK_STATUS_CHANGE_IND_T;
```

Link Status Change response

Variable	Type	Value / Range	Description
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F8B	HIL_LINK_STATUS_CHANGE_RES

Table 134: HIL_LINK_STATUS_CHANGE_RES_T – Link Status Change response

Packet structure reference

```
/* LINK STATUS CHANGE RESPONSE */
#define HIL_LINK_STATUS_CHANGE_RES         HIL_LINK_STATUS_CHANGE_IND+1

typedef HIL_PACKET_HEADER      HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F22	HIL_BUSSCAN_REQ
Data			
ulAction	uint32_t	0x01 0x02 0x03	Action to perform HIL_BUSSCAN_CMD_START HIL_BUSSCAN_CMD_STATUS HIL_BUSSCAN_CMD_ABORT

Table 135: HIL_BUSSCAN_REQ_T – Bus Scan request

Packet structure reference

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

#define HIL_BUSSCAN_CMD_START    0x01
#define HIL_BUSSCAN_CMD_STATUS   0x02
#define HIL_BUSSCAN_CMD_ABORT    0x03

typedef struct HIL_BUSSCAN_REQ_DATA_Ttag
{
  uint32_t ulAction;
} HIL_BUSSCAN_REQ_DATA_T;

typedef struct HIL_BUSSCAN_REQ_Ttag
{
  HIL_PACKET_HEADER       tHead;
  HIL_BUSSCAN_REQ_DATA_T  tData;
} HIL_BUSSCAN_REQ_T;
```

Bus Scan confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	12 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F23	HIL_BUSSCAN_CNF
Data			
ulMaxProgress	uint32_t		Maximum time (in milliseconds) application has to wait for scan to complete
ulActProgress	uint32_t		Already elapsed time (in milliseconds)
abDevice List[4]	uint8_t		List of available devices on the fieldbus system

Table 136: HIL_BUSSCAN_CNF_T – Bus Scan confirmation

Packet structure reference

```
/* BUS SCAN CONFIRMATION */
#define HIL_BUSSCAN_CNF                    HIL_BUSSCAN_REQ+1

typedef struct HIL_BUSSCAN_CNF_DATA_Ttag
{
  uint32_t ulMaxProgress;
  uint32_t ulActProgress;
  uint8_t  abDeviceList[4];
} HIL_BUSSCAN_CNF_DATA_T;

typedef struct HIL_BUSSCAN_CNF_Ttag
{
  HIL_PACKET_HEADER      tHead;
  HIL_BUSSCAN_CNF_DATA_T tData;
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulLen	uint32_t	4	Packet Data Length (in Bytes)
ulCmd	uint32_t	0x00002F24	HIL_GET_DEVICE_INFO_REQ
Data			
ulDeviceIdx	uint32_t	n	Fieldbus specific device identifier

Table 137: HIL_GET_DEVICE_INFO_REQ_T – Get Device Info request

Packet structure reference

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

typedef struct HIL_GET_DEVICE_INFO_REQ_DATA_Ttag
{
  uint32_t ulDeviceIdx;
} HIL_GET_DEVICE_INFO_REQ_DATA_T;

typedef struct HIL_GET_DEVICE_INFO_REQ_Ttag
{
  HIL_PACKET_HEADER              tHead;
  HIL_GET_DEVICE_INFO_REQ_DATA_T tData;
} HIL_GET_DEVICE_INFO_REQ_T;
```

Get Device Info confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	8 + n 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F25	HIL_GET_DEVICE_INFO_CNF
Data			
ulDeviceIdx	uint32_t	n	Identifier of device
ulStructId	uint32_t	m	Identifier of structure type
	Structure		Fieldbus specific data structure

Table 138: HIL_GET_DEVICE_INFO_CNF_T – Get Device Info confirmation

Packet structure reference

```
/* GET DEVICE INFO CONFIRMATION */
#define HIL_GET_DEVICE_INFO_CNF            HIL_GET_DEVICE_INFO_REQ+1

typedef struct HIL_GET_DEVICE_INFO_CNF_DATA_Ttag
{
  uint32_t ulDeviceIdx;
  uint32_t ulStructId;
  /* uint8_t  tStruct; Fieldbus specific structure */
} HIL_GET_DEVICE_INFO_CNF_DATA_T;

typedef struct HIL_GET_DEVICE_INFO_CNF_Ttag
{
  HIL_PACKET_HEADER              tHead;
  HIL_GET_DEVICE_INFO_CNF_DATA_T tData;
} HIL_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

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	uint32_t	0x00002F82	HIL_VERIFY_DATABASE_REQ

Table 139: HIL_VERIFY_DATABASE_REQ_T – Verify Database request

Packet structure reference

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

typedef struct HIL_VERIFY_DATABASE_REQ_Ttag
{
  HIL_PACKET_HEADER tHead;                    /* packet header */
} HIL_VERIFY_DATABASE_REQ_T;
```

Verify Database confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	 116 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F83	HIL_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.

Table 140: HIL_VERIFY_DATABASE_CNF_T – Verify Database confirmation

Packet structure reference

```
/* VERIFY DATABASE CONFIRMATION */
#define HIL_VERIFY_DATABASE_CNF            HIL_VERIFY_DATABASE_REQ+1

typedef struct HIL_VERIFY_SLAVE_DATABASE_LIST_Ttag
{
    uint32_t  ulLen;
    uint8_t   abData[16];
} HIL_VERIFY_SLAVE_DATABASE_LIST_T;

typedef struct HIL_VERIFY_MASTER_DATABASE_Ttag
{
  uint32_t  ulMasterSettings;   /* field bus independent changes */
  uint32_t  ulMasterStatus;     /* field bus specific status */
  uint32_t  ulReserved[2];
} HIL_VERIFY_MASTER_DATABASE_T;

#define HIL_CIR_MST_SET_STARTUP       0x00000001
#define HIL_CIR_MST_SET_WATCHDOG      0x00000002
#define HIL_CIR_MST_SET_STATUSOFFSET  0x00000004
#define HIL_CIR_MST_SET_BUSPARAMETER  0x00000008

typedef struct HIL_VERIFY_DATABASE_CNF_DATA_Ttag
{
    HIL_VERIFY_SLAVE_DATABASE_LIST_T               tNewSlaves;
    HIL_VERIFY_SLAVE_DATABASE_LIST_T               tDeactivatedSlaves;
    HIL_VERIFY_SLAVE_DATABASE_LIST_T               tChangedSlaves;
    HIL_VERIFY_SLAVE_DATABASE_LIST_T               tUnchangedSlaves;
    HIL_VERIFY_SLAVE_DATABASE_LIST_T               tImpossibleSlaveChanges;
    HIL_VERIFY_MASTER_DATABASE_T                   tMasterChanges;
} HIL_VERIFY_DATABASE_CNF_DATA_T;

typedef struct HIL_VERIFY_DATABASE_CNF_Ttag
{
  HIL_PACKET_HEADER                        tHead;    /* packet header  */
  HIL_VERIFY_DATABASE_CNF_DATA_T           tData;    /* packet data    */
} HIL_VERIFY_DATABASE_CNF_T;
```

5.9.2 Activate Configuration Database

This packet indicates the master to activate the new configuration.

Activate Database request

Variable	Type	Value / Range	Description
ulDest	uint32_t	0x00000020	HIL_PACKET_DEST_DEFAULT_CHANNEL
ulCmd	uint32_t	0x00002F84	HIL_ACTIVATE_DATABASE_REQ

Table 141: HIL_ACTIVATE_DATABASE_REQ_T – Activate Database request

Packet structure reference

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

typedef struct HIL_ACTIVATE_DATABASE_REQ_Ttag
{
  HIL_PACKET_HEADER tHead;                    /* packet header */
} HIL_ACTIVATE_DATABASE_REQ_T;
```

Activate Database confirmation

Variable	Type	Value / Range	Description
ulLen	uint32_t	16 0	Packet Data Length (in Bytes) If ulSta = HIL_S_OK Otherwise
ulSta	uint32_t	See below	Status / Error Code, see Section 6
ulCmd	uint32_t	0x00002F85	HIL_ACTIVATE_DATABASE_CNF
Data			
abSlvSt[16]	uint8_t	n	State of the Slaves after Configuration

Table 142: HIL_ACTIVATE_DATABASE_CNF_T – Activate Database confirmation

Packet structure reference

```
/* ACTIVATE DATABASE CONFIRMATION */
#define HIL_ACTIVATE_DATABASE_CNF          HIL_ACTIVATE_DATABASE_REQ+1

typedef struct HIL_ACTIVATE_DATABASE_CNF_DATA_Ttag
{
  uint8_t                abSlvSt[16];  /* State of the slaves after configuration */
} HIL_ACTIVATE_DATABASE_CNF_DATA_T;

typedef struct HIL_ACTIVATE_DATABASE_CNF_Ttag
{
  HIL_PACKET_HEADER                        tHead;  /* packet header */
  HIL_ACTIVATE_DATABASE_CNF_DATA_T         tData;
} HIL_ACTIVATE_DATABASE_CNF_T;
```

6 Status and error codes

The following status and error codes may be returned in *ulSta* 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	HIL_S_OK Success, Status Okay
0xC0000001	ERR_HIL_FAIL Fail
0xC0000002	ERR_HIL_UNEXPECTED Unexpected
0xC0000003	ERR_HIL_OUTOFMEMORY Out Of Memory
0xC0000004	ERR_HIL_UNKNOWN_COMMAND Unknown Command
0xC0000005	ERR_HIL_UNKNOWN_DESTINATION Unknown Destination
0xC0000006	ERR_HIL_UNKNOWN_DESTINATION_ID Unknown Destination ID
0xC0000007	ERR_HIL_INVALID_PACKET_LEN Invalid Packet Length
0xC0000008	ERR_HIL_INVALID_EXTENSION Invalid Extension
0xC0000009	ERR_HIL_INVALID_PARAMETER Invalid Parameter
0xC000000C	ERR_HIL_WATCHDOG_TIMEOUT Watchdog Timeout
0xC000000D	ERR_HIL_INVALID_LIST_TYPE Invalid List Type
0xC000000E	ERR_HIL_UNKNOWN_HANDLE Unknown Handle
0xC000000F	ERR_HIL_PACKET_OUT_OF_SEQ Out Of Sequence
0xC0000010	ERR_HIL_PACKET_OUT_OF_MEMORY Out Of Memory
0xC0000011	ERR_HIL_QUE_PACKETDONE Queue Packet Done
0xC0000012	ERR_HIL_QUE_SENDPACKET Queue Send Packet
0xC0000013	ERR_HIL_POOL_PACKET_GET Pool Packet Get
0xC0000015	ERR_HIL_POOL_GET_LOAD Pool Get Load
0xC000001A	ERR_HIL_REQUEST_RUNNING Request Already Running
0xC0000100	ERR_HIL_INIT_FAULT Initialization Fault
0xC0000101	ERR_HIL_DATABASE_ACCESS_FAILED Database Access Failed
0xC0000119	ERR_HIL_NOT_CONFIGURED Not Configured

Value	Definition / Description
0xC0000120	ERR_HIL_CONFIGURATION_FAULT Configuration Fault
0xC0000121	ERR_HIL_INCONSISTENT_DATA_SET Inconsistent Data Set
0xC0000122	ERR_HIL_DATA_SET_MISMATCH Data Set Mismatch
0xC0000123	ERR_HIL_INSUFFICIENT_LICENSE Insufficient License
0xC0000124	ERR_HIL_PARAMETER_ERROR Parameter Error
0xC0000125	ERR_HIL_INVALID_NETWORK_ADDRESS Invalid Network Address
0xC0000126	ERR_HIL_NO_SECURITY_MEMORY No Security Memory
0xC0000140	ERR_HIL_NETWORK_FAULT Network Fault
0xC0000141	ERR_HIL_CONNECTION_CLOSED Connection Closed
0xC0000142	ERR_HIL_CONNECTION_TIMEOUT Connection Timeout
0xC0000143	ERR_HIL_LONELY_NETWORK Lonely Network
0xC0000144	ERR_HIL_DUPLICATE_NODE Duplicate Node
0xC0000145	ERR_HIL_CABLE_DISCONNECT Cable Disconnected
0xC0000180	ERR_HIL_BUS_OFF Network Node Bus Off
0xC0000181	ERR_HIL_CONFIG_LOCKED Configuration Locked
0xC0000182	ERR_HIL_APPLICATION_NOT_READY Application Not Ready
0xC002000C	ERR_HIL_TIMER_APPL_PACKET_SENT Timer App Packet Sent
0xC02B0001	ERR_HIL_QUE_UNKNOWN Unknown Queue
0xC02B0002	ERR_HIL_QUE_INDEX_UNKNOWN Unknown Queue Index
0xC02B0003	ERR_HIL_TASK_UNKNOWN Unknown Task
0xC02B0004	ERR_HIL_TASK_INDEX_UNKNOWN Unknown Task Index
0xC02B0005	ERR_HIL_TASK_HANDLE_INVALID Invalid Task Handle
0xC02B0006	ERR_HIL_TASK_INFO_IDX_UNKNOWN Unknown Index
0xC02B0007	ERR_HIL_FILE_XFR_TYPE_INVALID Invalid Transfer Type
0xC02B0008	ERR_HIL_FILE_REQUEST_INCORRECT Invalid File Request
0xC02B000E	ERR_HIL_TASK_INVALID Invalid Task
0xC02B001D	ERR_HIL_SEC_FAILED Security EEPROM Access Failed

Value	Definition / Description
0xC02B001E	ERR_HIL_EEPROM_DISABLED EEPROM Disabled
0xC02B001F	ERR_HIL_INVALID_EXT Invalid Extension
0xC02B0020	ERR_HIL_SIZE_OUT_OF_RANGE Block Size Out Of Range
0xC02B0021	ERR_HIL_INVALID_CHANNEL Invalid Channel
0xC02B0022	ERR_HIL_INVALID_FILE_LEN Invalid File Length
0xC02B0023	ERR_HIL_INVALID_CHAR_FOUND Invalid Character Found
0xC02B0024	ERR_HIL_PACKET_OUT_OF_SEQ Packet Out Of Sequence
0xC02B0025	ERR_HIL_SEC_NOT_ALLOWED Not Allowed In Current State
0xC02B0026	ERR_HIL_SEC_INVALID_ZONE Security EEPROM Invalid Zone
0xC02B0028	ERR_HIL_SEC_EEPROM_NOT_AVAIL Security EEPROM Not Available
0xC02B0029	ERR_HIL_SEC_INVALID_CHECKSUM Security EEPROM Invalid Checksum
0xC02B002A	ERR_HIL_SEC_ZONE_NOT_WRITEABLE Security EEPROM Zone Not Writeable
0xC02B002B	ERR_HIL_SEC_READ_FAILED Security EEPROM Read Failed
0xC02B002C	ERR_HIL_SEC_WRITE_FAILED Security EEPROM Write Failed
0xC02B002D	ERR_HIL_SEC_ACCESS_DENIED Security EEPROM Access Denied
0xC02B002E	ERR_HIL_SEC_EEPROM_EMULATED Security EEPROM Emulated
0xC02B0038	ERR_HIL_INVALID_BLOCK Invalid Block
0xC02B0039	ERR_HIL_INVALID_STRUCT_NUMBER Invalid Structure Number
0xC02B4352	ERR_HIL_INVALID_CHECKSUM Invalid Checksum
0xC02B4B54	ERR_HIL_CONFIG_LOCKED Configuration Locked
0xC02B4D52	ERR_HIL_SEC_ZONE_NOT_READABLE Security EEPROM Zone Not Readable

Table 143: Status and error codes

7 Appendix

7.1 List of figures

7.2 List of tables

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.
69800 Saint Priest
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