Operating Instruction Manual

Tag List Editor

Viewing and Editing Tags in Firmware Files

V1.5

Hilscher Gesellschaft für Systemautomation mbH

www.hilscher.com

DOC110306OI08EN | Revision 8 | English | 2020-04 | Released | Public

Table of Contents

1 Introduction

1.1 About this Manual

This manual describes the

- Tag List Editor
- the command line tool makenxo
- the command line tool tagtool

The command line tools are included with the Tag List Editor.

This manual is aimed at Original Equipment Manufacturers (OEMs), who use Hilscher netX controllers in their hardware and who need to adapt the standard loadable firmware provided by Hilscher to their hardware.

1.1.1 List of Revisions

Index	Date	Version	Chapter	Revision
5	2015-02-05	1.2	-	Tag List Editor V1.2 released with same functionality as V1.1.
6	2018-06-06	1.3	1.3 5.1 5.1.9	Legal Notes updated New command for tagtool added: **dump_tagdefs** (write tag definitions to a file in JSON format)
7	2019-03-08	1.4	all 8.2	Document updated according to Tag List Editor version 1.4 (editor now supports NXI file types for netX 90 and netX 4000) Section *Structure of netX 90*/4000 Firmware File added
8	2020-04-03	1.5	all 1.1.3 1.2 5.1.10 8.2	Document updated according to Tag List Editor version 1.5 Section *Terms, Abbreviations and Definitions* expanded. Section *About Tag List Editor*: Table 2 added. Section *upd_extfw: Update Base and Extended Firmware Files* added. Section *Structure of netX 90/4000 Firmware File* expanded.

1.1.2 Conventions in this Manual

Operation instructions, a result of an operation step or notes are marked as follows:

Operation Instructions:

➢ <instruction>

or

1. <instruction>
2. <instruction>

Results:

➪ <result>

Notes:

Important: <important note>

Note: <note>

<note, where to find further information>

1.1.3 Terms, Abbreviations and Definitions

Term	Definition
APP CPU	CPU used for application firmware
BSL	Bootstrap Loader, also called Second Stage Loader. The newer versions from 1.3 and up are in NXF format.
COM CPU	CPU used for communication firmware
MXF	Maintenance Firmware file for netX 90 or netX 4000.
NAE, NAI, NXE, NXF, NXI, NXO	netX Firmware file. For a description of each file name extension and the relation to the netX chip type, see Table 2 on page 7.
OEM	Original Equipment Manufacturer

Table 1: Terms, Abbreviations and Definitions

1.2 About Tag List Editor

A tag list is a list of structured configuration parameters which has been compiled into an executable firmware file for a netX. The tag list contains parameters to configure the firmware. The information contained in the tag list (e.g. the location of the file system) must be available before any configuration files are accessed.

Firmware files also contain a data structure called the "device header", which contains additional information about the firmware and the device(s) the firmware is intended for. The device header can be viewed and edited in the Tag List Editor as well.

The Tag List Editor provides a graphical user interface to view and edit the tag list and the device header stored in a firmware file. Tag list and device header contents can also be loaded and saved as raw data.

Note, that the Tag List Editor is a tool for experts. Misusing this tool can have serious consequences for your device and/or the network in which the device is used.

The Tag List Editor allows you to:

- edit the tag list
- edit the device header
- construct an NXO file from its basic components.

If you are an OEM who uses a Hilscher netX chip, you can thus use the Tag List Editor to change certain parameters contained in the loadable standard firmware for the netX provided by Hilscher, in order to make it operable with your own hardware setup and in order to let you use your own customized product identification on the network bus.
For more information on adapting loadable standard firmware, see section *Customizing Hilscher Standard Firmware for OEM Devices* on page 18.

Also included in the Tag List Editor are two command line tools:

- makenxo combines a header binary, ELF file and tag list to an NXO file.
- tagtool allows you to print, compare (diff) and manipulate (patch) the tag list and device header of a firmware file. The tagtool can replace the tag list by a new tag list loaded from a binary file.

Note: The tag list editor and the command line tools work on files with common header V3.0 and device header V1.0.
Files containing a common header with a version below 3.0 are rejected.
Files with a common header with a version above 3.0 are accepted, but a warning is shown.

Table 2 lists the file name extensions with relation to the netX chip type.

netX chip type	File name extension	Description
netX 90/4000	NXI	netX Firmware file for the COM CPU of netX 90 or netX 4000.
	NXE	netX Firmware extension file for the COM CPU of netX 90.
	NAI	netX Firmware file for the APP CPU of netX 90 or netX 4000.
	NAE	netX Firmware extension file for the APP CPU of netX 90.
	MXF	Maintenance Firmware file for netX 90 or netX 4000.
netX 10/50/51/52/100/500	NXF	netX Firmware file for netX 10/50/51/52/100/500.
	NXO	netX option module, modular firmware for the netX 100/500.
	BSL	Bootstrap Loader, also called Second Stage Loader. The newer versions from 1.3 and up are in NXF format: Uses the NXF file format with a slightly different layout, where the tag list is located in front of the firmware code. The actual extension for Second Stage Loader files is ".bin".

Table 2: File name extension for netX Firmware (Overview)

1.2.1 System Requirements

- Windows® XP (32 bit),
  Windows 7 (32 or 64 bit),
  Windows 10 (32 or 64 bit)
- 30 MB free hard disk space

1.2.2 Limitations

- The Tag List Editor handles only firmware files which contain the Common Header V3.
- It can only edit existing tag lists. It is not possible to insert or remove tags. This is a deliberate limitation.
- The tag list may contain tags which are unknown to the editor. These tags are not displayed, but they are kept and written back when the file is saved.
- There is no automatic consistency checking. The Tag List Editor does not ensure that the settings you make are reasonably applicable to the used hardware and firmware. For instance, when editing task priorities, you can only set values which are valid as task priorities, but the editor does not check if all tasks have distinct priorities.

2 Installing Tag List Editor

Under Windows XP, you must be logged in as an administrator in order to install the Tag List Editor. It will, however, run under a limited user account.

Under Windows 7 or 10, it can be installed and run under a limited user account.

2.1 Installation Instructions

The installation is straightforward. The installer is a single executable file. Run it and follow the instructions:

1. On the welcome page, click on **Next**.
2. Read the license agreement. If you accept it, select **I accept the agreement** and click on **Next**. Otherwise, click on **Cancel** to abort the installation.
3. Select a directory for the installation, or accept the predefined one. Then, click on **Next**.
4. Accept the components to install. All components are necessary to use the tag list editor.

   Click on **Next** to continue.
5. Select whether you want to create a Start menu item, and what it should be called.

   Click on **Next** to continue.
6. On this page, you define whether you want to create a Start menu icon, a desktop icon and a quick launch icon.

   Also, select whether the environment variable "PATH_NXOEDITOR" should be set. This variable is necessary if you want to use the command line tools, makenxo and tagtool.

   Click on **Next** to continue.
7. You are now ready to install. Review the installation settings on this page and click on **Install** or click on **Back** to go back and change the installation settings.
8. The change log is shown. Click on **Next**
9. Click on **Finish** to exit the installer.

2.2 Configuration Files

There is a default configuration file which is installed with the Tag List Editor. When you close the Tag List Editor, any changes you have made are written to a personal configuration file which is created in your user directory. The settings stored in the personal configuration file override those in the default configuration.

Default configuration: <install directory>\application\Modulator.cfg

e.g. C:\Program Files\Hilscher GmbH\Tag List Editor\application\ Modulator.cfg

User configuration: <User local application data directory>\Modulator.cfg

e.g. C:\Documents and Settings\user\Local Settings\Application Data\ Modulator.cfg

The user configuration file is not deleted when you uninstall the Tag List Editor. If you encounter any problems after an update, delete this file manually. The file is located in a hidden directory; in order to make it visible, open the Explorer's folder options and select "Show hidden files and folders".

3 Functions of the Tag List Editor

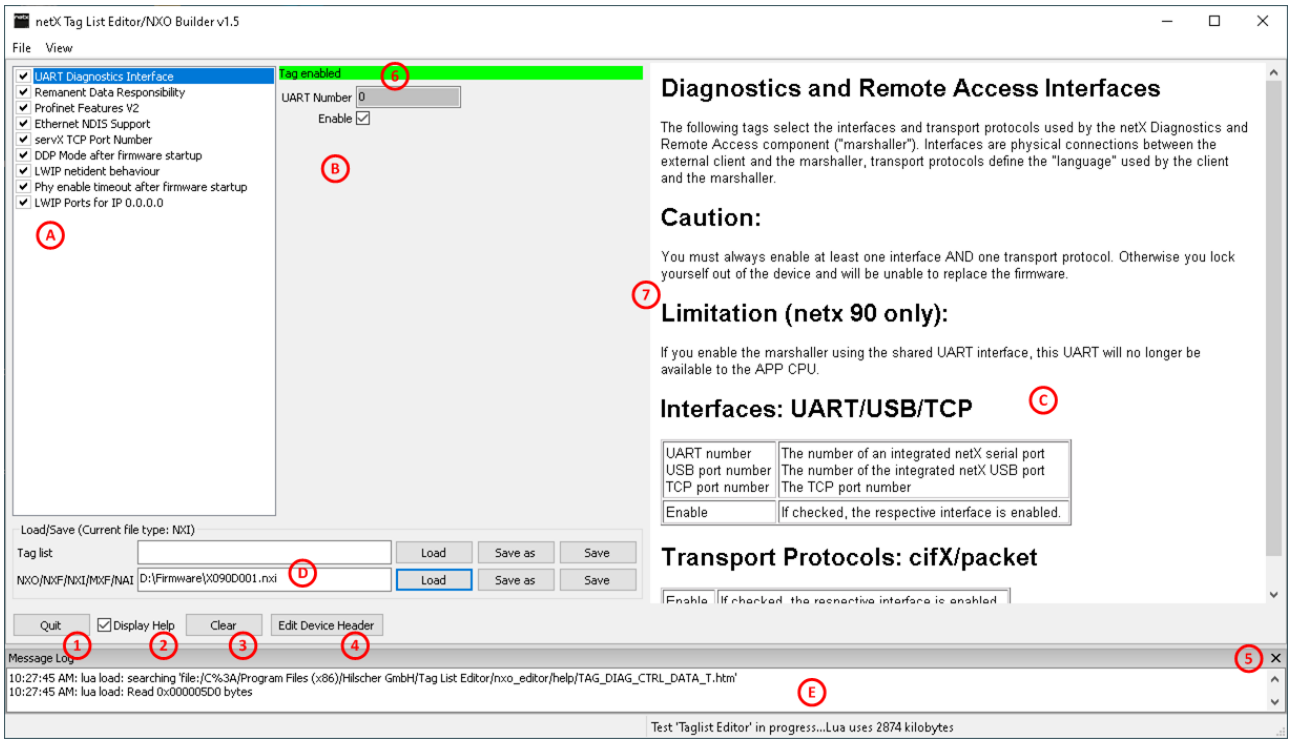

Figure 1: The Tag List Editor Window

Area	Description
A	The list of tags which are currently loaded and which are known by the editor. The check box next to each tag indicates if the tag is enabled.
B	The editing area for the currently selected tag.
C	The help page for the currently selected tag.
D	Load/save options.
E	The log area.

Table 3: Main Areas of the Editor Window

Control Element	Description
1	Exits the Tag List Editor.
2	Turns the help area on or off.
3	Resets the Tag List Editor to the initial state.
4	Opens a window which allows editing of the device header if the currently loaded file contains one.
5	Closes the log area. In order to display it again, select it in the View menu.
6	Indicates if the currently selected tag is enabled.
7	The edge between the editing area and the help area. Drag it to adjust the size of the help area.

Table 4: Control Elements of the Editor Window

3.1 Loading and Saving

The Load/Save area allows you to save and load complete firmware files, the tag list and the headers and/or ELF sections of an NXO file.

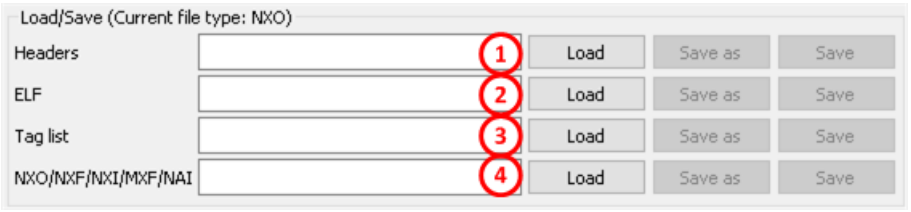

Figure 2: Load/Save Area configured for NXO Files

Figure 3: Load/Save Area configured for Firmware Files (1)

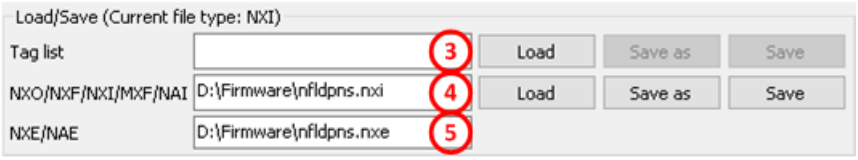

Figure 4: Load/Save Area configured for Firmware Files (2)

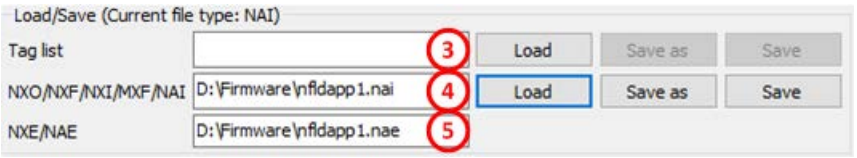

Figure 5: Load/Save Area configured for Firmware Files (3)

Control Element	Description
①	Load or save the header section of an NXO file.
②	Load or save the ELF section of an NXO file.
③	Load or save the tag list of a firmware file.
④	Load or save a firmware file.
⑤	Extension firmware file.

Table 5: Control Elements for Loading and Saving

Options ① and ② are shown initially, after the Clear button has been used to return to the initial state or after an NXO file has been loaded. They are hidden when a firmware of a different type is loaded.

In case you load a firmware file that requires an extension firmware file, a popup window will ask for the location of the extension firmware file. Options ⑤ is shown only if an extension firmware file has been loaded.

Loading files is always possible, saving only when the respective data is in memory. That is, for instance, after loading a firmware file containing a tag list you can save the firmware file or the tag list itself.

This allows for a number of use cases:

- Editing a tag list which is embedded in a file: Load a firmware file which contains a tag list, edit tags and save the file.
- Editing a tag list which is in a separate file: Load a tag list from a separate file, edit it and save the modified tag list.
- Extracting a tag list which is embedded in a file: Load an firmware file containing a tag list and save the tag list to a separate file.
- Inserting/replacing a tag list: Load an firmware file, then load a tag list from a separate file and save the modified firmware file.
- Constructing an NXO file: Load the Headers, ELF file and tag list from separate files and save them as an NXO file.

The loading functions perform some checks on the files. In order to understand the meaning and extent of the checks, some knowledge of the file format is required. See section *Structure of NXO/NXF File* on page 32 for a short description.

Note: Gap data following the sections of the file (headers, data/ELF, tag list) is kept as long as these sections are not replaced. If a section is replaced by loading a new header or ELF file, the gap data is discarded and replaced with 0 to 3 padding bytes.

Exception: If the tag list is located between the header and data sections as it is the case in the Second stage loader, the gap data is retained when the tag list is replaced, and shortened or extended to ensure that the data section following it stays at the same absolute location in the file.

3.1.1 Loading an NXO Header Binary

This function loads an NXO header binary. The file must contain the common header V3 and any following headers but not the initial default header.

The following checks are performed after loading the file. If any of them fails, the file is rejected:

- The common header must have major version 3
- The file size, ulHeaderLength and bNumModuleInfo must be consistent
- If a tag list is in memory and ulTagListSizeMax is set, the tag list must not be longer than ulTagListSizeMax.

3.1.2 Loading an NXO ELF File

This function loads the ELF section of an NXO file. If the file does not have the ELF signature, it is rejected.

3.1.3 Loading a Tag List

In the following cases, the file is rejected:

- The tag list ends within a tag
- The tag list has a 4-byte or 8-byte end marker followed by additional data
- The size information of a known tag is inconsistent with the structure definition in the editor
- A common header with the field ulTagListSizeMax greater than 0 is in memory and the tag list is longer than the maximum size
- A file with the tag list located before the data section is in memory and the tag list does not fit between the headers and the data section.

In the following case, a warning is shown:

- The tag list end does not have an end marker or only a 4-byte end marker. If the maximum tag list size allows it, the editor offers to replace the end-marker with an 8-byte end marker.

3.1.4 Loading a Firmware File

In the following cases, the file is rejected:

- none of the known file types can be identified
- the common header version is below 3
- the file structure cannot be recognized (e.g. because of invalid offset/size information in file headers)

In the following cases, the file is accepted but a warning message is shown:

- any of the checksums in the boot header/common header are incorrect
- the common header version is higher than 3.0
- the file contains data in-between the headers, data and tag list sections

If the checks on the file succeed, the checks for tag lists described in the previous section are performed.

3.1.5 Saving a Firmware File

The file is saved and the tag list, the common header fields pertaining to the tag list and the checksums are updated.

3.2 Editing Tags

3.2.1 Editing Fields

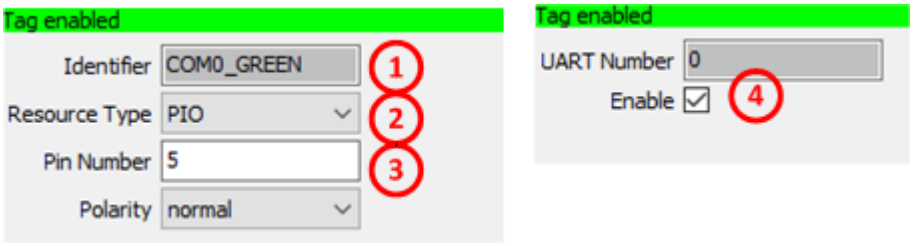

Figure 6: Editing Controls for Values in Tags

Each field in a tag is presented as a numeric field ③, a text field ①, a selection box ② or a checkbox ④.

Entries with a grey background ① are read-only. For instance, task priority tags contain pre-defined name entries which are used by the firmware to retrieve the tags.

Numeric values ③ are displayed either in decimal or hex notation, but may be entered in either notation.
To enter hexadecimal values, start with the prefix `0x` followed by `0…9` and `a…f` (use lower case letters).

The editor will prevent you from entering numbers which are larger than the bit width of the field. For some fields, a more limited value range is defined. You can always enter the value 0, because it is part of the prefix of hexadecimal numbers.

Note: Priorities and priority ranges

If you change task priority and task token settings, always set both to the same value.

Priority base and priority range values must fulfill the following condition:

base + range - 1 <= max. value

For instance, if the task priority range is 2, the base priority must be set to a value less than 55, since 55 is the highest value.

3.2.2 Disabling Tags

The checkbox to the left of each tag determines whether the tag is enabled, that is, visible to the firmware:

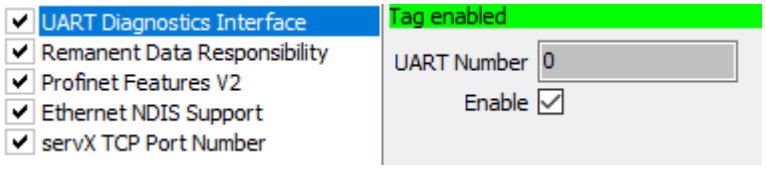

Figure 7: The selected Tag is enabled

Figure 8: The selected Tag is disabled

Important:

Disabling a tag makes it invisible to the firmware. It does NOT necessarily disable the feature which the tag configures. The firmware may have a built-in default configuration in which the feature is enabled. In this case, the feature will be active and using the default configuration.

If a tag contains an on/off option, the secure way to disable the feature is to enable the tag and set the option to "off".

3.3 Editing Device Header

When a file containing a device header is in memory, the **Edit Device Header** button is enabled. It opens a window which allows you to view and edit the device header.

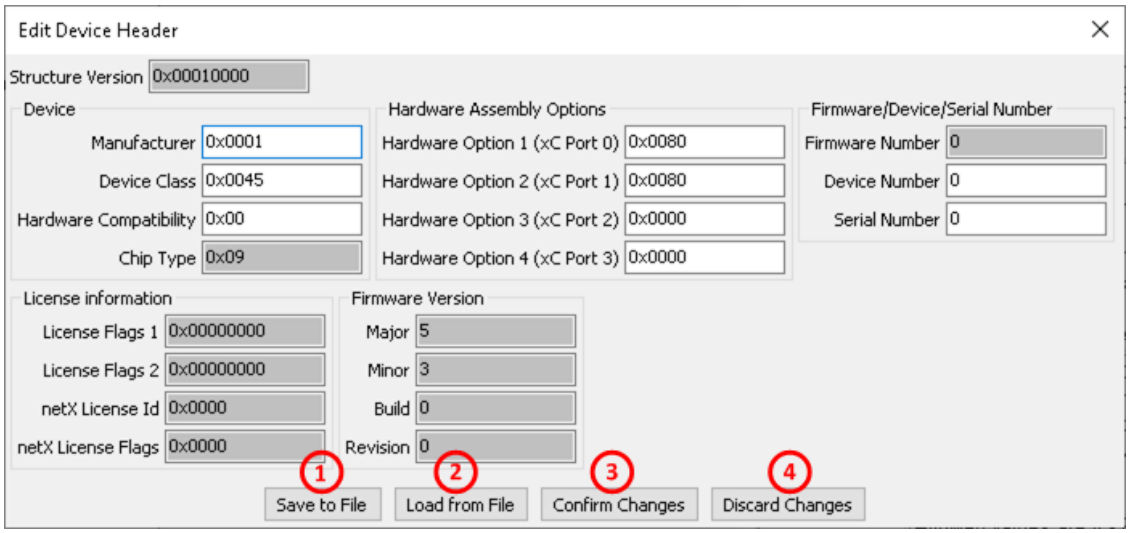

Figure 9: The Device Header Editor Window

Manufacturer code, device class, hardware options, device product code and device serial number can be edited. All other fields are read-only. You will find a description of the editable fields in *Table 7: Parameters in Firmware Header to be Adapted by OEM* on page *19.*

Control Element	Description
①	Writes the device header with the currently displayed values to a binary file.
②	Loads the device header from a binary file. The file is only accepted if the size and version entry match those of the device header V1. If the file is accepted, the contents are displayed in the editing window, but not replaced in the loaded NXF/NXI/NXO file until you exit the window using **Confirm Changes**.
③	Stores any changes you have made and closes the window
④	Discards any changes you have made and closes the window. This is equivalent to clicking on the close icon.

Table 6: Control Elements of the Device Header Editor Window

4 Customizing Hilscher Standard Firmware for OEM Devices

4.1 Adapting Device Header

If you are an OEM who uses a Hilscher netX chip, you probably need to adapt certain parameters in the header of the standard loadable firmware (NXF file) provided by Hilscher for the netX, in order to make the firmware operable with your hardware setup.

The following parameters need to be checked (and changed, if necessary):

- Manufacturer Code ①
- Device Class ②
- Hardware Compatibility Index ③
- Hardware Assembly Options of xC Ports ④

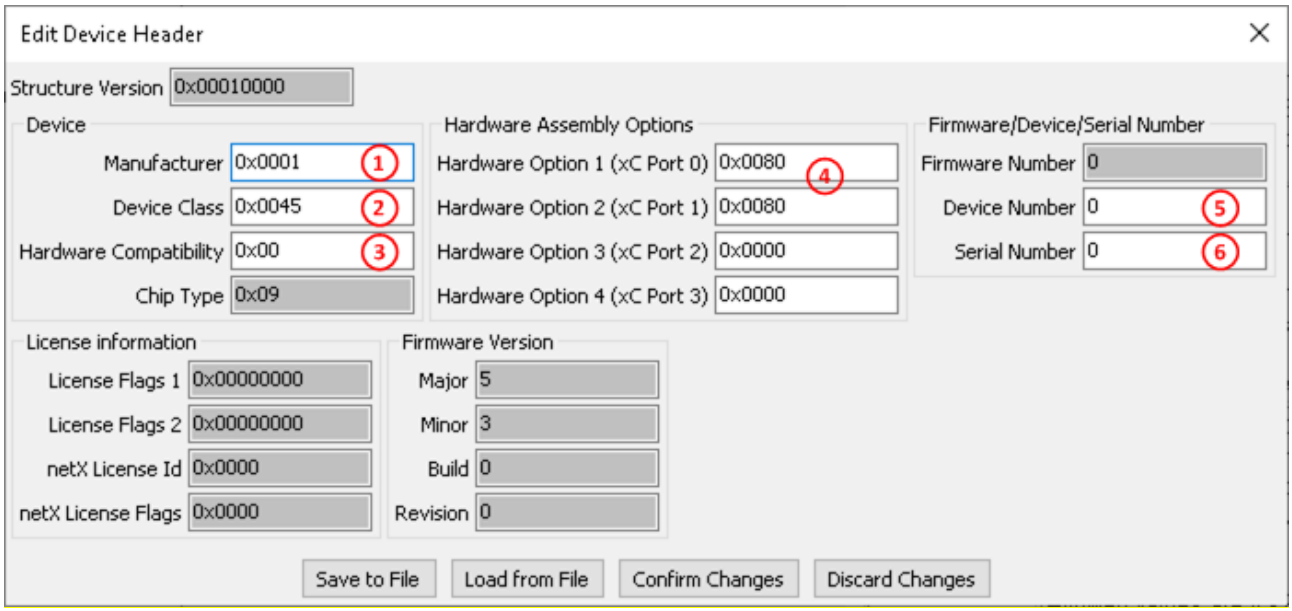

Figure 10: Parameters in Firmware Header to be Adapted by OEM

Pos.	Field/Parameter	Description
①	Manufacturer	Enter here your OEM code, which has been assigned to you by Hilscher. By default, this field is filled with the code for Hilscher (0x0001).
②	Device Class	Enter here the code for the device class (netX chip) you are using. If you are not using your own coding (only in agreement with Hilscher GmbH), use the applicable standard code: netX 10: 0x001A netX 50: 0x000C netX 51: 0x0026 netX 52: 0x002D netX 90 (COM CPU): 0x003C netX 90 (COM CPU) with SDRAM at host interface: 0x0045 netX 100: 0x0007 netX 500: 0x0002 netX 4000 (COM CPU) without SDRAM: 0x003A netX 4000 (COM CPU) with SDRAM at host interface: 0x0043 netX 4000 (COM CPU) with SDRAM at memory interface: 0x0044

Pos.	Field/Parameter	Description
③	Hardware Compatibility	The hardware compatibility is an index which indicates whether the firmware is compatible with the hardware version. (The index starts with zero and is incremented every time changes to the hardware require changes to the firmware)
④	Hardware Assembly Options	Enter here the values that determine the hardware configuration of the xC ports. The xC ports connect the netX to its network/bus. The **Hardware Option 1** field represents **xC Port 0** of the netX, etc: Hardware Option 1: xC Port 0 Hardware Option 2: xC Port 1 Hardware Option 3: xC Port 2 Hardware Option 4: xC Port 3 For a list of values which can be entered here (e. g. 0x0090 = Serial Peripheral Interface), please refer to the *Hardware Assembly Options* table in the *System Information Block* section of the *Dual-Port Memory Interface Manual*. The availability of the xC Ports depends on the device class of the netX.
⑤	Device Number	Preset to 0. Reserved for future use.
⑥	Serial Number	Preset to 0. Reserved for future use.

Table 7: Parameters in Firmware Header to be Adapted by OEM

If you – before you proceed to edit the device header – want to check the corresponding values of the hardware parameters (like e. g. the hardware compatibility index) stored in the security memory, you can do so by using the **Identify Hardware Request** function in SYCON.net or in the cifX Test Application. For more information, see section *Identifying netX Hardware Through Packets* in the *Dual-Port Memory Interface Manual*, DOC060302DPMxxEN.

Note: Before the firmware can be downloaded to the OEM hardware, the configuration software (SYCON.net, Communication Studio or netX Configuration Tool) compares the values of the parameters stored in the firmware header with the values of the corresponding parameters stored in the security memory of the netX chip. The aim of the comparison is to find out whether the firmware file is appropriate for the connected hardware. If the values are not identical, the validation fails and the firmware can not be downloaded or updated in the device.

A validation also takes place before the firmware is started up in the hardware each time after the OEM hardware has been switched on.
Here, the values of the header parameters are checked by the Second Stage Boot Loader against the values of the corresponding parameters stored in the security memory. If the values in the header do not match the values in the security memory, the firmware will not be started.

4.2 Adapting Tags for Product Identification on the Network Bus

This section is valid for firmware files in NXF format only.

If you as an OEM want your device to identify itself on the network bus with your own manufacturer/vendor or device labeling, you need to change the tags for the product identification in the loadable firmware accordingly. Product identification parameters consist in most cases at least of a vendor ID and a device ID. Vendor IDs are defined and allocated by the organization responsible for the management of the bus protocol. So, for example, if you are using PROFINET, you need to contact PROFIBUS International and ask them to assign you a vendor identification. After this vendor identification has been assigned to you, you can use the Tag List Editor to set the product identification parameters in the firmware. In case of PROFINET the product identification parameters also consist of a Device ID, which can be assigned by the OEM.

Important: After you have changed the product identification in the firmware file, you also need to adapt the product identification parameters in the corresponding device description file.

Example: Adapting Tags for Product Identification in a Loadable Firmware for a PROFINET Network

The following step by step instructions provide an example of how to edit product identification tags in a loadable firmware, in order to use an own customized product identification on a network bus. This is an example for PROFINET, note that other protocols may use other tags.

IMPORTANT:

- Use the Tag List Editor responsibly.
- Misusing this tool can have serious consequences for your device and/or the network in which the device is used.
- Do not choose random settings for the product identification on the network bus.
- The user of this tool is responsible for setting the right parameters.

1. Start the Tag List Editor.
2. Load the firmware file.

➢ In the **Load/Save** area, click **Load** button next to the **NXO/NXF** field.

↬ The **Select NXO/NXF/bin file** dialog opens.

➢ Select the firmware (NXF) file, then click **Open** button.

➮ The firmware file is loaded in the Tag List Editor.

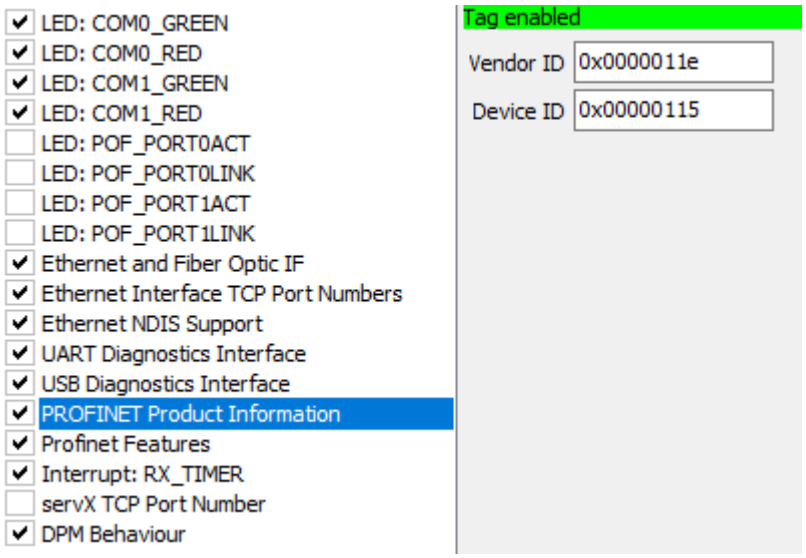

Figure 11: Adapting Product ID Tags in PROFINET Firmware

3. Edit the product identification tags.

- In the **List** area, select the **PROFINET Product Information** entry. Activate the check box in front of the entry in order to enable editing the tag (tag editing is disabled by default).
- In the **Editing** area, enter the Device ID and the Vendor ID assigned to you by the organization dealing with the allocation of Vendor IDs for this protocol (in case of PROFINET the PROFIBUS International Nutzerorganisation e. V.)

4. Save the changes to the firmware file.

- In the **Load/Save** area, click **Save** button next to the **NXO/NXF/…** field. (If you want to save the firmware file under a different name, click **Save as**.)
- Answer the **Overwrite file ?** question with **Yes**.

You have changed the product identification tags in the loadable firmware for a PROFINET IO Device.

Important: You also need to adapt the product identification parameters in the device description file of the PROFINET IO Device after you have changed the firmware file. For this, expert knowledge is required.

5 Functions of the Tagtool

Tagtool is a command line tool to perform operations on the tag list and device header of an firmware file. It can print, compare and modify the tag list and device header and replace the tag list.

Path

Tagtool is located in

<Install directory>\nxo_editor\tagtool.bat

The install directory is stored in the variable PATH_NXOEDITOR, unless you disable this option in the installer. In order to call the tool, add %PATH_NXOEDITOR%\nxo_editor to the search path.

5.1 Using the Tagtool

The general usage is

```
tagtool <command> [flags] <arguments>
```

with the following commands, flags and argument types:

<command>	<arguments>	Function
settags	input_file taglist_file output_file input_file = input firmware file taglist_file = binary file containing a tag list output_file = output firmware file	Replace the tag list
list	input_file input_file = input firmware file	Print tag list and device header
diff	input_file1 input_file2 input_file1/2 = input firmware files	Compare tag lists and device headers
edit	input_file patch_file output_file input_file = input firmware file patch_file = text file describing modifications output_file = output firmware file	Apply patch to tag list and device header
dump_tagdefs	output_file	Write the tag definitions to a file in JSON format
upd_extfw	input_base_file input_extension_file output_base_file output_extension_file	Copy the device and module info headers of the base firmware into the extension firmware, and update all checksums in both output files.
help -h		Print usage information
help_tags		Print the known tags
help_const		Print the known value constants
-version		Print version information

Table 8: Tagtool Commands and Arguments

Flag	Function
-v	enables verbose output
-debug	enables debug output

Table 9: Tagtool Flags

Since the tool is wrapped in a batch file, you must use following syntax to call it from another batch file:

```
call tagtool <command> [flags] <arguments>
```

5.1.1 help, -h, /?: Print Usage Information

```
tagtool help
tagtool –h
tagtool /?
tagtool
```

This command prints the help page.

5.1.2 help_tags: Print known Tags

```
tagtool help_tags
```

This command prints a list of the known tags.

5.1.3 help_const: Print known Constants

```
tagtool help_const
```

This command prints a list of the known value constants. These constants may be used in a patch file instead of values.

5.1.4 –version: Print version Information

```
tagtool -version
```

This command prints version information.

5.1.5 settags: Replace the Tag List

```
tagtool settags input_file taglist_file output_file
```

This command reads input_file, replaces the tag list with the contents of taglist_file, updates the offset/size fields of the common header and the checksums and writes the result to output_file.

Example:

Assume you have saved a tag list as tags_xc3_bin using the editor. The following command will put this tag list into an NXO file:

```
>tagtool settags nx100mpi.nxo tags_xc3.bin nx100mpi_xc3.nxo
```

5.1.6 list: Print the Tag List and Device Header

```
tagtool list input_file
```

This command reads input_file and prints the tag list and device header. For unknown tags, only the tag ID is listed.

Example:

```
>tagtool list nx100mpi.nxo
Tag 1: RCX_TAG_LED (0x00001040)
ENABLED
.szIdentifier       = FB_STA_RED
.ulUsesResourceType = 0x00000002
.ulPinNumber        = 0x00000005
.ulPolarity         = 0x00000000


Tag 2: RCX_TAG_LED (0x00001040)
ENABLED
.szIdentifier       = FB_STA_GREEN
.ulUsesResourceType = 0x00000002
.ulPinNumber        = 0x00000004
.ulPolarity         = 0x00000000


Tag 3: RCX_TAG_XC (0x00001050)
ENABLED
.szIdentifier = PB_XC
.ulXcId       = 0x00000002

...
# Tag 7: unknown (0x00000806)


DEVICE_HEADER_V1_T
.ulStructVersion      = 0x00010000
.usManufacturer       = 0x0001
.usDeviceClass        = 0x0007
.bHwCompatibility     = 0x00
.bChipType            = 0x00
.usReserved           = 0x0000
.usHWOptions_1        = 0x0000
.usHWOptions_2        = 0x0000
.usHWOptions_3        = 0x0050
.usHWOptions_4        = 0x0000
.ulLicenseFlags1      = 0x00000000
.ulLicenseFlags2      = 0x00000000
.usNetXLicenseID      = 0x0000
.usNetXLicenseFlags   = 0x0000
.ulFwVersion_Major    = 0x0002
.ulFwVersion_Minor    = 0x0004
.ulFwVersion_Build    = 0x0001
.ulFwVersion_Revision = 0x0000
.ulFwNumber           = 0x00000000
.ulDeviceNumber       = 0x00000000
.ulSerialNumber       = 0x00000000
.ulReserved1          = 0x00000000
.ulReserved2          = 0x00000000
.ulReserved3          = 0x00000000
```

5.1.7 diff: Compare Tag Lists and Device Headers of two Files

```
tagtool diff input_file1 input_file2
```

This command reads input_file1 and input_file2, compares the tag lists and device headers and prints output in the patch file format which can be read by the edit command.

Note: The tag lists of the two files must contain exactly the same tags in the same order.

For any tags whose values differ, the tag from the second file is printed as an edit record which can be used as input for the "edit" function.

Example:

The file nx100mpi.nxo was edited to use xC unit 3 and saved as nx100mpi_xc3.nxo.

```
>tagtool diff nx100mpi.nxo nx100mpi_xc3.nxo
Tag 3: RCX_TAG_XC (0x00001050)
    ENABLED
    .szIdentifier = PB_XC
SET .ulXcId       = 0x00000003
```

5.1.8 edit: Patch Tag List and Device Header

```
tagtool edit input_file patch_file output_file
```

This command reads input_file, applies the modifications listed in patch_file to the tag list and device header, updates the checksums and writes the result to output_file.

Example:

Assume you have edited a firmware file in the editor and saved the result to a different file.

Using the diff and edit commands, it is possible to extract the changes and apply them to another file, for example, a new version of the firmware.

```
>tagtool diff nx100mpi.nxo nx100mpi_xc3.nxo >diff_xc3.txt
>tagtool edit nx100mpi_new.nxo diff_xc3.txt nx100mpi_new_xc3.nxo
```

5.1.9 dump_tagdefs: Write the tag definitions to a file

```
tagtool dump_tagdefs output_file
```

This command writes the tag definitions to the output file in JSON format.

Example:

```
>tagtool dump_tagdefs tagdefs.json
```

5.1.10 upd_extfw: Update Base and Extended Firmware Files

A netX 90 firmware file is typically stored in the internal Flash memory of the netX 90. In case, the firmware is larger than the ssize of the internal Flash memory, the firmware is split into two files:

- a base firmware file (NXI or NAI) which fits into the internal Flash memory and
- an extension firmware file (NXE or NAE) which is stored in the external Flash memory.

The base and extension firmware files share the same device and module info headers.

They also share a checksum entry in the common header, which is calculated from both files.

These shared data structures signify that the base and extension firmware file are compatible.

The firmware files has to be patched in two steps:

- Use the edit command to patch the tag list or device header of the base firmware.
- Use the upd_extfw command on the base and the extension firmware file.

The upd_extfw command copies the device and module info headers of the base firmware to the extension firmware file. Then, it computes the shared checksum and stores it in both files. It will also update all checksums in both files.

5.2 Patch files

5.2.1 File Format

The patch file is a text file generated by the "diff" command and used as input by the "edit" command. It contains one or more edit records (patch records) which describe a patch to a data structure, either a tag or the device header. Each edit record is matched against all tags and the device header. If it matches exactly one data structure, the patch is applied. Otherwise, an error is raised.

The first line of an edit record specifies the type of data structure which this record should be matched to, either a particular tag type or the device header:

Structure type constraints	Function
TAG_NAME	Matches a tag of type TAG_NAME
Tag <ignored until colon>: TAG_NAME <The rest of the line is ignored>	Matches a tag of type TAG_NAME
DEVICE_HEADER_V1_T	Matches the device header

Table 10: Data Type Selection in an Edit Record

The following lines specify further constraints on the selected data structure:

Member constraints	Function
ENABLED	Matches a tag which is enabled
DISABLED	Matches a tag which is disabled
member_name = value	Matches a structure whose member has the given value

Table 11: Value Constraints in an Edit Record

Any lines which start with the SET keyword specify the actual patch. They are applied if a matching data structure has been found:

Member modifiers	Function
SET ENABLED	Enables the selected tag
SET DISABLED	Disables the selected tag
SET member_name = value	Assigns a value to a member

Table 12: Value Assignment in an Edit Record

String values may be written with or without double quotes.

Numeric values may be written in decimal or hexadecimal notation or using the symbolic names shown by the help_const command.

Byte arrays are written as a list of decimal or hexadecimal values, separated by comma and/or space, optionally enclosed in curly braces:

```
SET abEraseChipCmd = {0xc7, 0x94, 0x80, 0x9a}
SET abEraseChipCmd = 0xc7 0x94 0x80 0x9a
SET abEraseChipCmd = 199, 148, 128, 154
```

Empty lines are ignored, and everything following a # until the end of the line is ignored.

Example:

```
Tag 15: RCX_MOD_TAG_IT_XC (0x00001050)
.szIdentifier = "RTE_XC1"
DISABLED
SET .ulXcId = 2
SET ENABLED
```

This selects a tag with type RCX_MOD_TAG_IT_XC which has the identifier string "RTE_XC1" and is currently disabled. If the tag list contains exactly one tag which matches this description, it is enabled and its ulXcId field is set to 2.

Note: Some tags contain nested structures. To access these values, the full path from the root of the structure must be specified:

```
Tag 2: TAG_BSL_HIF_PARAMS (0x40000001)
.ulBusType = 0x00000000
.tDpmIsaAuto.ulIfConf0    = 0x00000000
.tDpmIsaAuto.ulIfConf1    = 0x00000000
.tDpmIsaAuto.ulIoRegMode0 = 0x00000001
.tDpmIsaAuto.ulIoRegMode1 = 0x00000002
.tPci.bEnablePin  = 0x01
.tPci.bPinType    = 0x01
.tPci.bInvert     = 0x80
.tPci.usPinNumber = 0x000f
```

Note: In some tags, two values are combined in one byte, word or dword. In the textual representation used by "list", "diff" and "edit", these values appear as separate values.

For instance, in the following tag:

```
Tag 5: TAG_BSL_USB_PARAMS (0x40000004)
.bEnable        = 0x01
.bPullupPinType = 0x01
.bInvert        = 0x80
.usPullupPinIdx = 0x0001
```

bPullupPinType and bInvert share one byte in the binary encoding.

bPullupPinType occupies bits 0-6 (value range 0x00-0x7f) and

bInvert occupies bit 7 (value is either 0x00 or 0x80).

This is currently the case with the following tags:

TAG_BSL_HIF_PARAMS_DATA_T

TAG_BSL_SDMMC_PARAMS_DATA_T

TAG_BSL_USB_PARAMS_DATA_T

TAG_BSL_FSU_PARAMS_DATA_T

5.2.2 How to generate a Patch File

Method 1: If you have an original firmware and a copy of the file with a modified tag list/device header, use the "diff" command to extract the changes.

Method 2: The output of the "list" function can be parsed by the "edit" function.

Redirect the output to a text file and open this file in an editor. Change the values as desired and add the keyword "SET" at the beginning of every line which contains a change.

Method 3: Write it manually according to the format described above.

5.2.3 Error Messages pertaining to the Patch File

Error message	Meaning
parse error	The format of a line could not be recognized.
parse error (no tag specified)	The file must start with a line which selects a tag or the device header
unknown type name <type name>	You have misspelt the tag name, or the program does not know this tag.
type <type name> has no member <member name>	The type is known, but the type does not have a member with the given name.
tried to get member of primitive type <type name>	The path specified in a value condition or assignment is too deeply nested.
Failed to parse value in match/assignment: type=<type name> value=<value>	The value does not have the correct type or cannot be parsed. If you specified a value constant, it may be spelt incorrectly.
edit record matches no tag	The tag list contains no tag which has the correct type and satisfies the conditions specified in the edit record.
edit record matches multiple tags	There is more than one tag which has the correct type and satisfies the conditions specified in the edit record.
Error while deserializing device header	
Device header has the wrong version: <32 bit version number>	Device header has the wrong version: <32 bit version number>
device header does not match	An edit record for the device header does not match
The file does not contain a tag list.	
BUG…	Any error messages starting with "BUG" indicate bugs in the program or the tag structure definitions. If you have made any changes of your own, check them. If you have not made any such changes, contact netX support.

Table 13: Error Messages which can occur when processing a Patch File

6 Functions of the MakeNXO Tool

MakeNXO is a command line tool to create an rcX loadable module in NXO format.

Path

MakeNXO is located in

<Install directory>\nxo_editor\makenxo.bat

The install directory is stored in the variable PATH_NXOEDITOR, unless you disable this option in the installer. In order to call the tool, add %PATH_NXOEDITOR%\nxo_editor to the search path.

6.1 Using MakeNXO

The call syntax is

```
makenxo –o output.nxo –H header_file [-t taglist_file] [-v] elf_file
```

- header_file is a binary file starting with a common header V3, usually followed by a device info block and a module info block.
- taglist_file is a binary file containing the tag list. The tag list is optional.
- elf_file is the ELF file of the loadable module
- output.nxo is the output file

The –v flag enables verbose output.

Example:

- If the firmware build process generates the following files:
- Fileheader.bin (HIL_FILE_STRIPPEDFIRMWARE_HEADER)
- Firmware.elf
- DefaultTagList.bin

the following command line will generate the nxo file:

```
makenxo -o Firmware.nxo -h Fileheader.bin -t DefaultTagList.bin
Firmware.elf
```

or, from a batch file:

```
call makenxo.bat -o Firmware.nxo -h Fileheader.bin -t
DefaultTagList.bin Firmware.elf
```

7 Lists

7.1 List of Figures

7.2 List of Tables

8 Appendix

8.1 Structure of NXO/NXF File

Files for netX 10/50/51/52/100/500 using the Common Header V3 have the following basic layout:

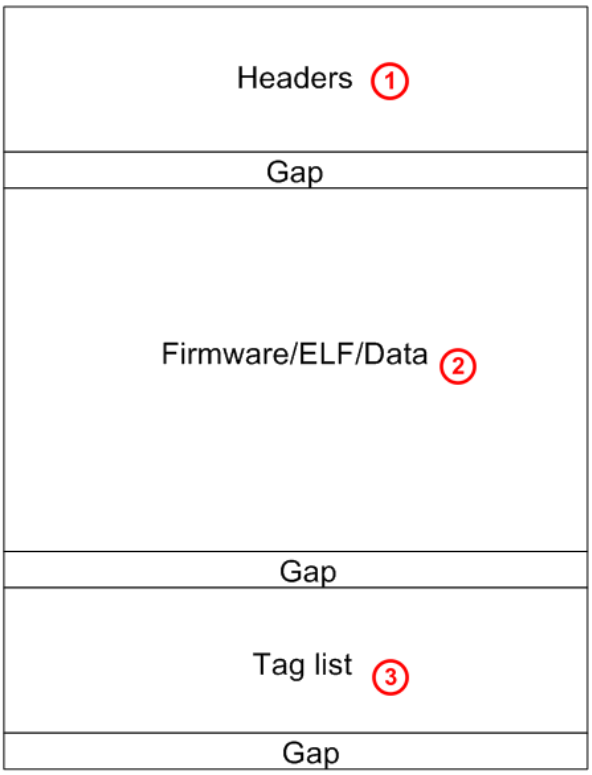

Figure 12: NXF/NXO file layout

1. The header section consists of a default or boot header, the common header, a device header and a number of module info headers.
   The Tag List Editor reads the common header for identification. The Tag List Editor can also edit the common header.
2. The data section: In an NXF file, this is the executable firmware, in an NXO file, it is an ELF.
3. The tag list is optional and may be located at the end of the file or between headers and data.
   It can be edited with the Tag List Editor.

There may be gaps between these sections, which are either padding or other data.

8.2 Structure of netX 90/4000 Firmware File

NXI, NXE, and MXF files for netX 90 and netX 4000 executed on the COM CPU have the following basic layout:

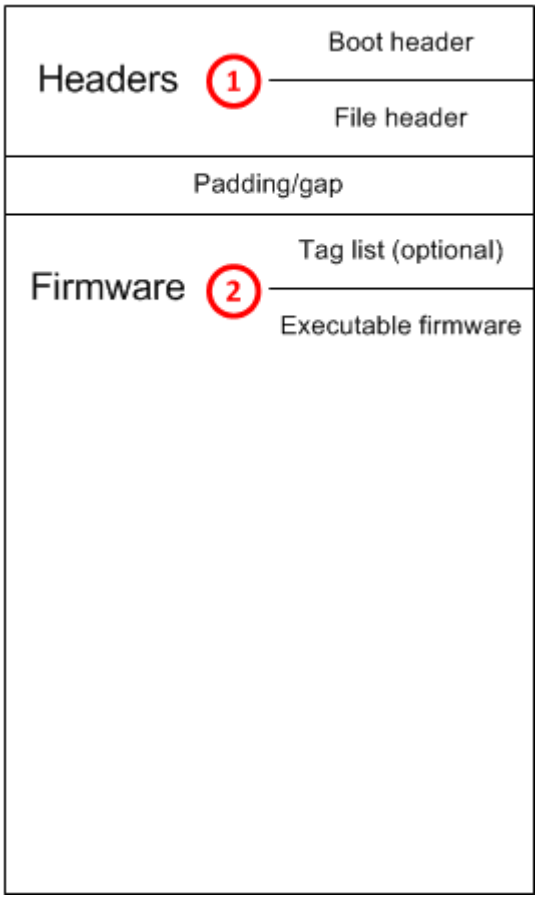

Figure 13: NXI, NXE, and MXF file layout

1. The header section consists of a boot header and a file header. The Tag List Editor reads the file header for identification. The Tag List Editor can also edit the file header.
2. The firmware section consists of the Tag List (optional) and the executable firmware. The Tag List can be edited with the Tag List Editor.

NAI and NAE for netX 90 executed on the APP CPU have the following basic layout:

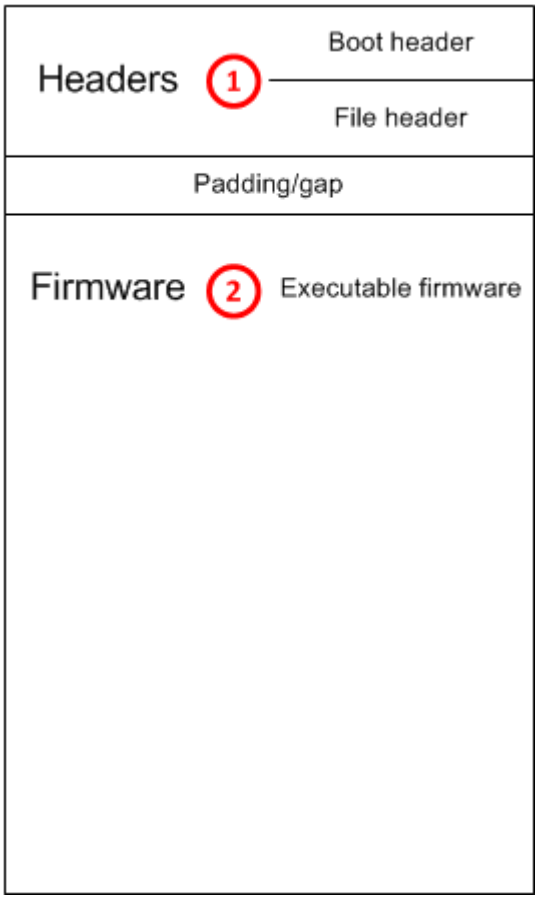

Figure 14: NAI and NAE file layout

1. The header section consists of a boot header and a file header. The Tag List Editor reads the file header for identification. The Tag List Editor can also edit the file header.
2. The firmware section consists of the executable firmware.

8.3 Legal Notes

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 fusion 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.

Registered Trademarks

Windows® XP, Windows® 7, and Windows® 10 are registered trademarks of Microsoft Corporation.

All other mentioned trademarks are property of their respective legal owners.

8.4 Contacts

Headquarters

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

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

Subsidiaries

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

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

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

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

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

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

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

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

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

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

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

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

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

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