optimize!
softing

User Guide

commScripter MBP

Version: EN-122022-3.60

Disclaimer of liability

The information contained in these instructions corresponds to the technical status at the time of printing of it and is passed on with the best of our knowledge. Softing does not warrant that this document is error free. The information in these instructions is in no event a basis for warranty claims or contractual agreements concerning the described products, and may especially not be deemed as warranty concerning the quality and durability pursuant to Sec. 443 German Civil Code. We reserve the right to make any alterations or improvements to these instructions without prior notice. The actual design of products may deviate from the information contained in the instructions if technical alterations and product improvements so require.

Trademarks

FOUNDATION™ and HART® are marks of the FieldComm Group of Austin, Texas, USA.
Modbus® is a registered trademark of Schneider Electric, licensed to the Modbus Organization, Inc.

OpenSource

To comply with international software licensing terms, we offer the source files of open source software used in our products. For details see https://opensource.softing.com/

If you are interested in our source modifications and sources used, please contact: info@softing.com

Softing Industrial Automation GmbH

Richard-Reitzner-Allee 6
85540 Haar / Germany
https://industrial.softing.com

+ 49 89 4 56 56-340

info.automation@softing.com
support.automation@softing.com

https://industrial.softing.com/support/support-form

Scan the QR code to find the latest documentation on the product web page under Downloads.

Table of Contents

This page is intentionally left blank.

1 About this guide

1.1 Read me first

Please read this guide carefully before using the commScripter MBP tool to ensure safe and proper use. Softing does not assume any liability for damages due to improper installation or operation of this product.

This document is not warranted to be error-free. The information contained in this document is subject to change without prior notice. To obtain the most current version of this guide, visit the download center on our website at: https://industrial.softing.com/en/downloads

1.2 Target audience

This document is addressed to field device developers and software engineers with a good working knowledge of HART/Modbus and FF or PROFIBUS PA technology and HART device descriptions to ensure that the processes and details described in this document are fully understood.

1.3 Typographic conventions

The following typographic conventions are used throughout Softing customer documentation:

Keys, buttons, menu items, commands and other elements involving user interaction are set in bold font and menu sequences are separated by an arrow	Open **Start → Control Panel → Programs**
Buttons from the user interface are enclosed in brackets and set to bold typeface	Press **[Start]** to start the application
Coding samples, file extracts and screen output is set in Courier font type	`MaxDlsapAddressSupported=23`
Filenames and directories are written in italic	Device description files are located in *C:\<Application name>\delivery\software\Device Description files*

CAUTION

CAUTION indicates a potentially hazardous situation which, if not avoided, may result in minor or moderate injury.

Note

This symbol is used to call attention to notable information that should be followed during installation, use, or servicing of this device.

1.4 Related documentation

User Guides:

- deviceUpdater-PA
- deviceUpdater-FF
- Field Device Software
- commModule MBP

1.5 Document feedback

We would like to encourage you to provide feedback and comments to help us improve the documentation. You can write your comments and suggestions to the PDF file using the editing tool in Adobe Reader and email your feedback to support.automation@softing.com.

If you prefer to write your feedback directly as an email, please include the following information with your comments:

- document name
- document version (as shown on cover page)
- page number

1.6 Release history

Product version	Modifications compared to previous version
3.10	Initial version
3.20	▪ Support for Modbus application devices added ▪ Control of power supply for application device via commScript supported
3.30	▪ Support for Profibus-PA in commKit approach added ▪ Definition of unmapped parameters in commScript supported
3.40	▪ Extend commScripter support for custom hardware (= non commModule based) projects ▪ Support for CS, AR, IS, SC Function Blocks in FF commKit devices added ▪ Improved PA diagnosis mapping and DIAG_EVENT_SWITCH handling ▪ Definition of storage type N for parameters supported ▪ Improved application interface for custom device implementation (i.e. non HART or Modbus mapped devices)
3.50	▪ NV-RAM data verification on commModule added ▪ Diagnosis mapping rearranged and unified ▪ Support for unmapped block parameters added, which can be written with the contained value ▪ PA: Totalizer to list of supported Function Blocks added ▪ FF: Support for SCP (Standard Connection Points) added ▪ FF: Support for Integrator function block added
3.60	▪ Added synchronous read mechanism ▪ Support for parameterized HART commands added ▪ Firmware update via fieldbus ▪ Optimized application power supply ▪ FF: Support for Output Splitter function block added

2 About commScripter MBP

Upgrading field devices with HART or Modbus interface for FOUNDATION Fieldbus or PROFIBUS PA means investing in both hardware and software development. The function block based object models of FOUNDATION Fieldbus and PROFIBUS PA require a mapping of the device parameters to the standardized function block application. Depending on the complexity of a field device this might prove to be a tough job.

With the introduction of the commModule MBP and the commScripter, Softing provides an easy way to enable HART or Modbus field devices for FOUNDATION Fieldbus or PROFIBUS PA.

The commModule MBP is a small fieldbus hardware interface for FOUNDATION Fieldbus and PROFIBUS PA. It comes with complete fieldbus stacks for both protocols and a configurable application layer allowing for a script-controlled mapping of the HART Commands or Modbus registers of existing field devices to the object models of FOUNDATION Fieldbus or PROFIBUS PA. MBP stands for “Manchester Bus-powered”, the Physical Layer both fieldbus protocols share.

The commScripter is a PC-based software development tool designed to create mapping information for the commModule MBP. It is a script compiler converting a mapping script to a binary S-Record file (.mot) that is downloaded (flashed) to the commModule MBP using the Renesas Flash Programmer (PC tool) and the Renesas E1 or E2 Lite Emulator (USB interface).

In summary, using the commScripter MBP and the commModule MBP you can create in a few steps an FF or PROFIBUS PA device from an existing HART or Modbus device:

1. First, a commScript is created to define how HART and Modbus device parameters are mapped to FF or PA block parameters.
2. When this is done, the Softing commScripter converts the commScript with the parameter mapping into an S-Record file.
3. Finally, the S-Record file is flashed into the commModule MPB using the Renesas Flash Programmer software and E1 or E2 Lite emulator.

In addition, the commScript will create standard description files according to the protocol used. For FF, a CFF file is generated. For PA a GSD file can be generated.

2.1 Scope of delivery

Download the zip file from product page to your PC and unpack the the included files to a directory of your choice. Excute start.exe in the selected directory to browse the content of the archive. The following software components and documents are included.

- commModule data sheet and HW manual
- Renesas Flash Programmer projects for downloading firmware and commScript tables to the commModule
- commScripter incl. batch files and manual
- demo device with simple mapping for HART (dev 0320) incl. commScript sources, FF DD, PROFIBUS EDD and GSD
- complex mapping for HART (dev 0307) incl. commScript sources, FF DD and PROFIBUS EDD and GSD
- complex mapping for Modbus (dev 0309) incl. commScript sources, FF DD and PROFIBUS EDD and GSD
- Softing License Manager for activating the commScripter license (setup file)
- genDomain incl. sample .ini files

For detailed information about the software components see Chapter Deliverables 113.

2.2 License

commScripter is a protected software and requires a license for usage. Depending on the fieldbus protocol and the protocol of the target device, Softing offers four different types of licenses, which may exist in parallel:

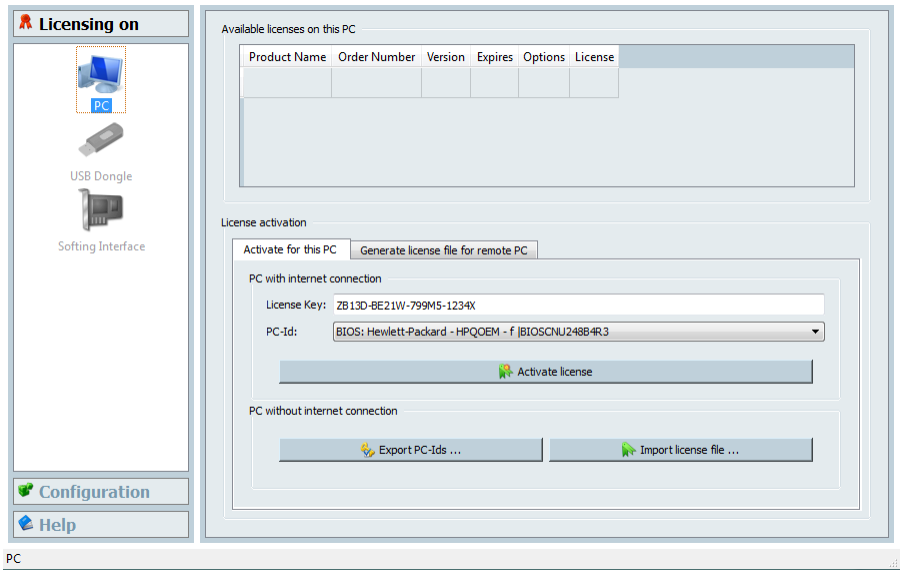
- LDA-KM-022451 License for enabling HART devices for FOUNDATION Fieldbus
- LDA-LM-022452 License for enabling HART devices for PROFIBUS PA
- LDA-KS-022453 License for enabling Modbus devices for FOUNDATION Fieldbus
- LDA-LS-022454 License for enabling Modbus devices for PROFIBUS PA

The licenses are single seat developer licenses and are either tied to a PC or a to a USB hardlock. In you opt for USB hardlock the license can be used on different PCs. If you want to use the USB hardlock you will need to order it separately.

1. Use the Softing License Manager that comes with commScripter for installing the license. An internet connection is required.
2. Start the Softing License Manager (Windows Startmenu Softing → License Manager → License Manager V4)

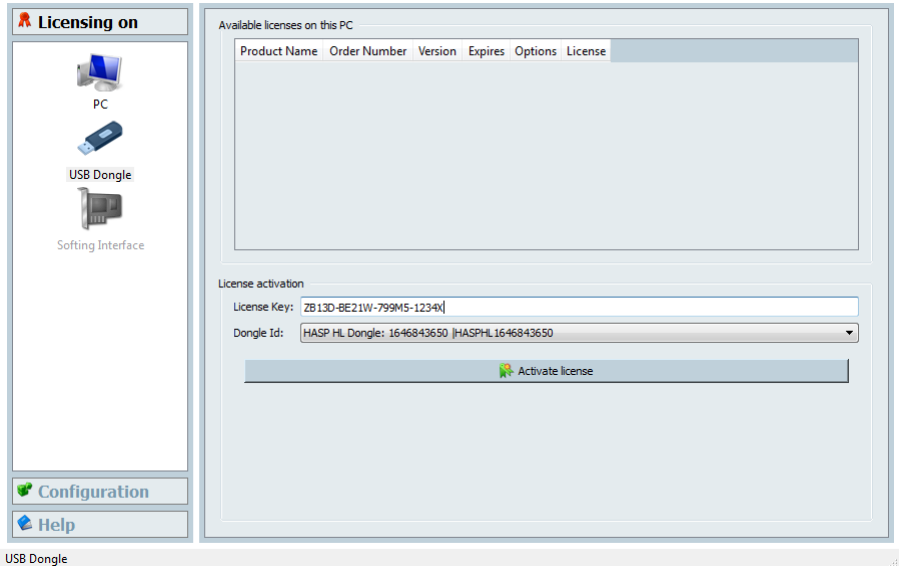
Binding the commScripter License to your PC:

3. Select the PC icon in the left column.
4. Enter the license key in the **License key** input field .
5. Click **[Activate license]**.

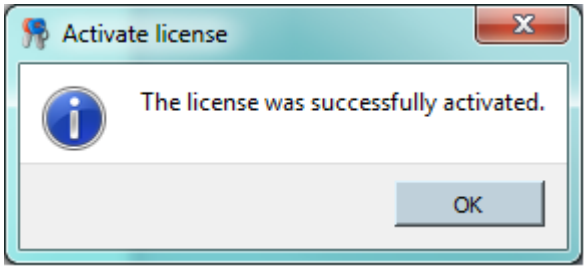


Binding the commScripter license to a USB hardlock “Sentinel MAC DL”:

6. Plug the hardlock into a USB port on your PC.
7. Wait until the hardlock is detected and installed.
8. Click the refresh icon in the top right corner of the license manager and select the USB hardlock icon in the left column.
9. Enter the license key in the **License key** input field .
10. Click **[Activate license]**.



After activating the license successfully, the following message appears:



2.3 System requirements

Hardware

- PC

Operating system

- Windows 10 (32 bit or 64 bit)

2.4 Glossary

Capability File (CF)
File describing the communication capabilities of a FOUNDATION Fieldbus device using the capability file format (CFF). In earlier specifications it was called “common file format”.

commDevice
Ready-to-use commModule with fieldbus stack, function block library, HART or Modbus Master, mapping application and commScript configuration table, acting as gateway between fieldbus and HART/Modbus device.

commKit
Collective term for hardware components and tools needed for developing configuration tables for the commModule MBP and flashing them into the commModule MBP.

commModule MBP
Fieldbus interface (hardware) for MBP physical layer. Contains protocol stacks for FOUNDATION Fieldbus H1 and PROFIBUS PA and configurable mapping application. Communicates with field device via serial interface using HART or Modbus commands.

commModule MBP Evaluation Kit
commModule MBP in a housing with screw terminals for fieldbus and serial interface and connector for the Renesas emulator. The evaluation kit comes with a configuration table that reads the identification and the primary value of a HART device. It is intended to be used for development and evaluation.

commScript
Text file describing the mapping of a specific HART or Modbus device to the function block application model of a FOUNDATION Fieldbus H1 or PROFIBUS PA field device. Read or write services on the fieldbus are translated into HART commands or Modbus services. The PC tool commScripter translates (compiles) the commScript into the binary commScript Configuration Table, which controls the mapping application in the commModule.

commScripter
PC tool for compiling a commScript into a binary file (commScript Configuration Table) to be downloaded to the commModule MBP.

commScript Configuration Table
Binary file containing the mapping information that controls the mapping between fieldbus services and HART/Modbus commands in a commModule MBP. The commScript Configuration Table is created by the commScripter and stored in the FLASH memory of the commModule.

Electronic Device Description (EDD)
Text file written in Device Description Language (DDL) describing the parameters, communication, user interfaces and operations of a field device. FOUNDATION Fieldbus EDDs are compiled into a binary format by means of a Tokenizer available from the FieldCommGroup.

FOUNDATION Fieldbus (FF)
Fieldbus for Process Industries specified and maintained by FieldComm Group.

Fieldbus Message Specification (FMS)

FF Spezification of the Layer 7 object model.

Function block

Function blocks represent the standardized application functions of a field device. There are function blocks for input, output and control like Analog Input, Digital Output or PID controller. Function blocks have a generic behavior, e. g. the functionality and the parameters of an analog input block are fixed and independent from the field device where it is implemented.

GenDomain

Tool for generation of download firmware file for SW download

MBP

Manchester Bus-powered Physical Layer as specified in IEC 61158-2 Type 1. It is used by both FOUNDATION Fieldbus H1 and PROFIBUS PA and is suited for intrinsic safety and for powering the field devices over the fieldbus.

PROFIBUS PA

PROFIBUS PA is the PROFIBUS application profile for process industries.

Resource/Physical Block

Each field device has exactly one resource block (FOUNDATION Fieldbus) or physical block (PROFIBUS PA) containing some basic information about the field device like manufacturer ID, device ID, serial number etc.

Renesas Flash Programmer

The Renesas Flash Programmer is a software tool and provides support for programming the on-chip flash memory of Renesas microcontrollers in each phase of development and mass production. It requires a Renesas E1 or E2 Lite emulator as interface between PC and commModule MBP Evaluation Kit. commKit contains project files supporting E1 emulator and project files supporting E2 Lite.

Renesas E1 Emulator

The Renesas E1 emulator is an on-chip debugging emulator and flash programmer that is suited for the RX64M microcontroller on the commModule MBP.

Renesas E2 Lite Emulator

The Renesas E2 Lite emulator is an on-chip debugging emulator and flash programmer that is suited for the RX64M microcontroller on the commModule MBP.

Transducer block

Transducer blocks are partially standardized but extendable and contain all parameters which are specific to a certain field device. They reflect the measurement principle (e.g. temperature, flow, pressure, level) and contain additional parameters for diagnostic functions, display etc. Transducer blocks produce process values for input function blocks or consume process values provided by output function blocks.

Tokenizer

A Tokenizer is needed for converting the EDD source files of a FOUNDATION Fieldbus device into a binary format that usually comes with a field device and that is used by engineering tools. The commScripter requires a symbol file generated by the Tokenizer for creating the commScript configuration table.

3 commDevices

commModule MBP is "single chip style" fieldbus interface for integrating FOUNDATION Fieldbus or PROFIBUS PA into field devices that support already HART or Modbus. Figure 1 shows the hardware components of commModule MBP. The fieldbus provides a supply voltage in the range of 9 to 32 V. The current consumption of commModule MBP can be adjusted by software in the range between 10 mA and 26 mA. As the medium attachment unit requires 10 mA for the signal modulation, up to 16 mA are left for the field device. Two different supply voltages (3.15 V and 6.2 V) are available, resulting in a maximum power budget of 90 mW for the field device. A programming and debugging interface allows for firmware and configuration download. A serial communication interface is used for communicating with the HART or Modbus device. The FSK modem (HART) or the RS485 transceiver (Modbus) in the field device has to be bypassed. That saves energy and allows transmission rates up to 115.2 kBit/s.

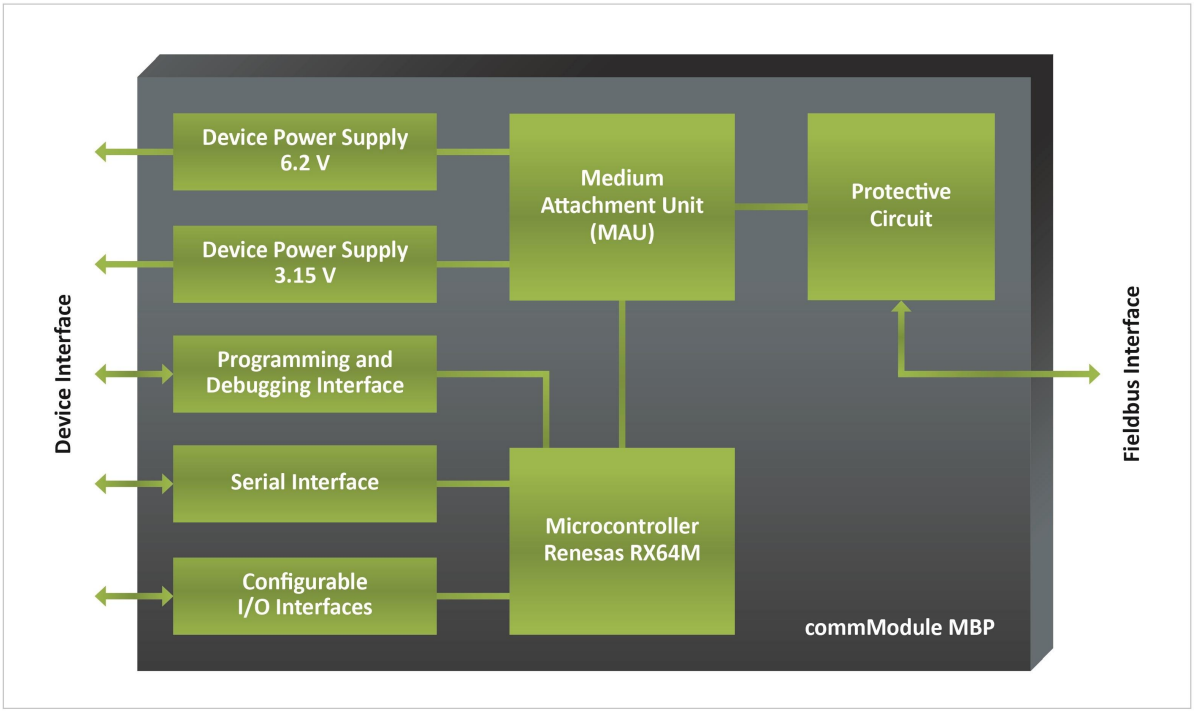


Figure 1: Hardware components of commModule MBP

commModule comes with a set of software components (see Figure 2). Fieldbus communication is handled by either a FOUNDATION Fieldbus or a PROFIBUS PA stack. Both are present, the selection is done by a hardware signal. A fieldbus function block application implements the required function blocks according to the device functionality. A HART master stack and a Modbus master stack are used for communication with the field device. A configurable mapping application forms the glue between the fieldbus stack and the HART/Modbus stack.

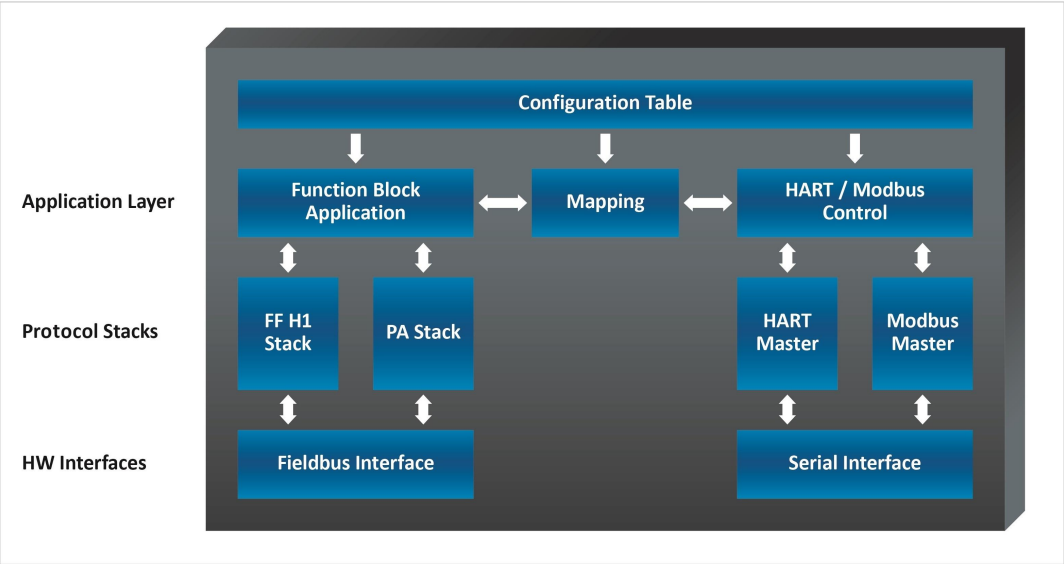


Figure 2: Software components of commModule MBP

Why do we need a configurable application? The HART Universal Commands and Common Practice Commands define a standardized set of access functions to basic device data. Access to the remaining data is done by using proprietary User Defined Commands. With Modbus there is no common definition about the usage of Modbus registers in field devices. Therefore mapping rules have to exist for each parameter in the function block application. Those mapping rules define what HART command or Modbus register has to be used to access the related data in the field device and how the response shall be interpreted. The definition of those mapping tools is done by creating a device-specific mapping script (commScript). The commScripter translates the script into a binary configuration table which can be downloaded into the commModule.

FOUNDATION Fieldbus and PROFIBUS PA are using similar models for the function block application. There are three categories of function blocks:

- Each field device has exactly one resource block (FOUNDATION Fieldbus) or physical block (PROFIBUS PA) containing some basic information about the field device like manufacturer ID, device ID, serial number etc.
- Function blocks represent the standardized application functions of a field device. There are function blocks for input, output and control like Analog Input, Digital Output or PID controller. Function blocks have a generic behavior, e. g. the functionality and the parameters of an analog input block are fixed and independent from the field device where it is implemented.
- Transducer blocks are partially standardized but extendable and contain all parameters which are specific to a certain field device. They reflect the measurement principle (e.g. temperature, flow, pressure, level) and contain additional parameters for diagnostic functions, display etc. Transducer blocks produce process values for input function blocks or consume process values provided by output function blocks.

The set of all function blocks on the commModule is called commDevice. Depending on the device function, a commDevice can consist of the following blocks:

- one FF resource block or PROFIBUS PA physical block
- an SCP (Standardized Connection Points) transducer block for FF devices
- a diagnostic transducer block
- up to six device-specific transducer blocks
- up to 20 function blocks

3.1 FF resource block

The commDevice resource block has 72 mandatory parameters (see Appendix 136). Up to 100 device-specific parameters can be added.

3.2 PROFIBUS PA physical block

The commDevice physical block has 38 mandatory parameters (see Appendix 144). Up to 100 device-specific parameters can be added.

3.3 FF SCP transducer block

The SCP transducer block is an integral part of the commDevice firmware. It is used to configure a device implementing the Standardized Connection Point concept. The SCP block is a special form of the transducer block class.

It is not directly associated with any physical transducer. The SCP block rather represents the SCP behavior of the device. (See also Chapter SCP in the related Field Device Software User Guide.)

3.4 Diagnostic transducer block

The diagnostic transducer block is an integral part of the commDevice firmware. The FF diagnostic transducer block has 37 parameters. The PROFIBUS PA diagnostic transducer block has 24 parameters. Device-specific modifications are not possible.

3.5 Demo transducer block

The Softing Demo device contains a transducer block which can generate several process values for demonstration purposes. When a channel is mapped accordingly, the device delivers live process values of predictable and configurable behavior, even if no HART device is connected. The FF Demo transducer block contains 51 parameters, the PROFIBUS PA Demo transducer block 45 parameters. This transducer block is only available in the Softing Demo device. It is recommended to remove the demo transducer block from all customer application devices.

3.6 Device-specific transducer blocks

Each FF transducer block has 14 mandatory parameters (see Appendix 140). Each PROFIBUS PA transducer block has 8 mandatory parameters (see Appendix 146). Up to 200 device-specific parameters can be added.

3.7 Function blocks

The commModule supports the following function block types:

- Analog Input function block
- Analog Output function block
- Discrete Input function block
- Discrete Output function block
- PID control function block (for FF devices only)
- Arithmetic function block (for FF devices only)
- Control Selector function block (for FF devices only)
- Input selector function block (for FF devices only)
- Output Splitter function block (for FF devices only)
- Signal characterizer function block (for FF devices only)
- Integrator function block (max. 1; for FF devices only)
- Totalizer function block (max. 2; for PA devices only)

3.8 Order of blocks

3.8.1 Order of blocks for FF

For FF devices the blocks will be created by commScripter in the following order:

1. Resource Block
2. Analog Input FBs
3. Discrete Input FBs
4. Analog Output FBs
5. Discrete Output FBs
6. PID control FBs
7. Integrator FB
8. Arithmetic FBs
9. Output Splitter FBs
10. Signal Characterizer FBs
11. Control Selector FBs
12. SCP TB
13. Softing DIAG TB
14. Demo TB (only in Softing Demo device)
15. Vendor TBs

The number of a specific FB type is defined in the commScript and can be 0 - 20. For Integrator FB, a maximum of 1 block can be defined.

The commScripter generates a CFF file that will contain the blocks and their indices so that the order of blocks also can been seen in this file. In addition commScripter generates output files ff_dev_<devType>.txt and ff_val_<devType>.txt (devType is the value set for the attribute DeviceType in the commScript). These files provide a description of the blocks and their parameters and values that are created by commScripter and also indicate which blocks are assigned to which indices.

3.8.2 Order of Blocks for PA

For PA devices the blocks will be created by commScripter in the following order.

1. Physical BLock
2. Analog Input FBs
3. Discrete Input FBs
4. Analog Output FBs
5. Discrete Output FBs
6. Totalizer FB
7. Demo TB (only on Softing Demo device)
8. Vendor TBs
9. Softing DIAG TB

The number of specific FBs is defined in the commScript and can be 0 - 20. Physical block is in slot 0, the first function block in slot 1 and the first transducer block will follow in the next slot after the last function block. For TOT-FB, a maximum of 2 blocks can be defined.

The commScripter generates output files pa_dev_<ManufIdent>.txt and ff_val_<ManufIdent>.txt (ManufIdent is the value set for the attribute ManufacIdentNumber in the commScript). These files provide a description of the blocks and their parameters and values that are created by commScripter. They also show the slot numbers for all blocks and the indices of their parameters and views.

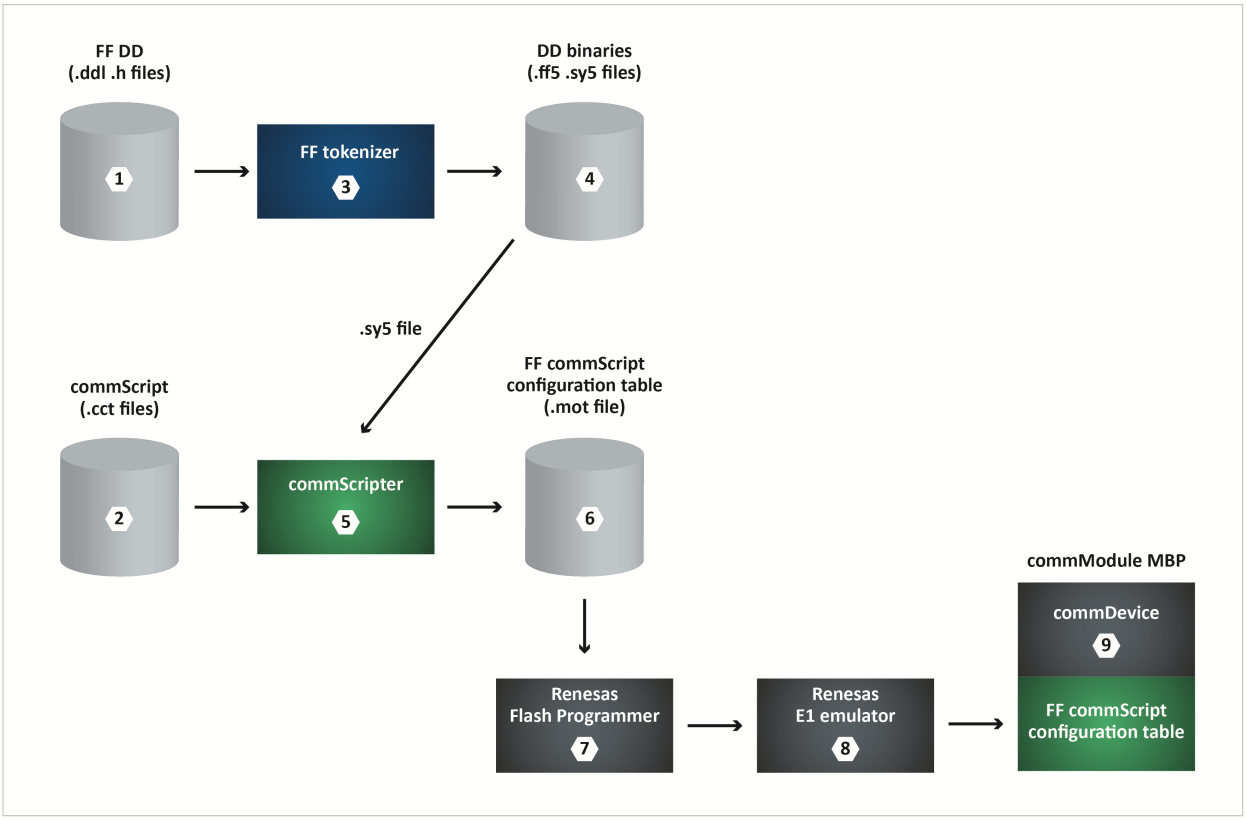
4 commScripter MBP workflow

4.1 Workflow FF

For building your commDevice you have to create an FF device description file (.ddl.h) ❶ and a commScript (.cct) file ❷ defining the mapping of HART or Modbus parameters to FF block parameters.

With the FF Tokenizer* ❸, the FF DD source files are converted into a binary file (.ff5) and a symbol file (.sy5 file) ❹. The symbol file and your commScript file are processed by the commScripter tool ❺. The commScripter generates an S-Record (.mot) file ❻ containing the FF commScript configuration table.

The FF commScript configuration table is flashed to the commModule MBP ❾ using the Renesas Flash Programmer software ❼ and the Renesas type E1 or E2 Lite emulator ❽ as shown in the following diagram:



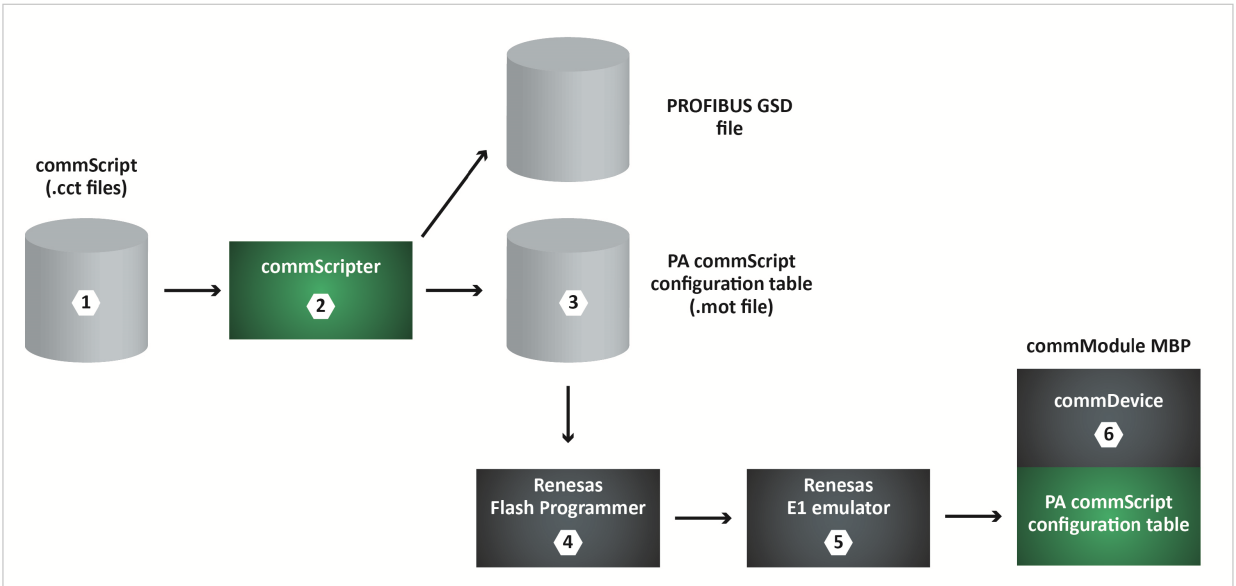
* The FF Tokenizer is the main component of the *DD Integrated Development Environment* (DD-IDE), a product offered by the FieldComm Group.

4.2 Workflow PROFIBUS PA

For building your commDevice you have to create a commScript (.cct) file ❶ defining the mapping of HART or Modbus parameters to PROFIBUS PA block parameters.

Your commScript file is then processed by the commScripter tool ❷ which generates an S-Record (.mot) file ❸ containing the PA commScript configuration table.

The PA commScript configuration table is flashed to the commModule MBP ❻ using the Renesas Flash Programmer ❹ and the Renesas E1 or E2 Lite emulator ❺ as shown in the following diagram:



4.3 Creating FF device descriptions

All blocks of an FF device have to be defined in an FF device description file (.ddl.h). See ❶ in the FF workflow.

The commScripter delivery package includes FF device descriptions for four example devices:

- DD\cs_start_hart_0320 DD source files for a demo device with a simple HART mapping
- DD\cs_hart_0307 DD source files for a complex HART device
- DD\cs_modb_0309 DD source files for a complex Modbus device

You can use one of the examples as a starting point for your device description.

Main description files

The example device descriptions have the following main DD files:

- cs_start_hart.ddl
- cs_hart.ddl
- cs_modbus.ddl

The main DD files include all files for defining function blocks, resource block and transducer blocks. The main DD file is passed as argument when invoking the FF Tokenizer.

Mandatory commDevice parameters

The resource block and the device-specific transducer blocks have a set of parameters which are mandatory for commDevices. The delivery package comes with a description of all these parameters.

Device-specific parameters

Device-specific parameters can be added to both the resource block and the device-specific transducer blocks.

4.3.1 Device identification

In the example device descriptions the device is identified by macros which are defined in header files.

For the start_HART example see the *cs_start_hart.h* file below:

```
/*
************************************************************
Device Identification
************************************************************
*/
 #define MANUFACTURER_ID_VALUE   0x1E6D11
 #define MANUFACTURER_ID_LABEL   "|en|Softing"

 #define DEVICE_TYPE_VALUE       0x0320
 #define DEVICE_TYPE_LABEL       "|en|CS Start HART"

 #define DEVICE_REVISION_VALUE   2

 #define DD_REVISION_VALUE       1

 #define NR_OF_SCP_BLOCKS        8
```

These macros are applied in the main DD file, ff_standard_param.ddl and cs_scp_block.ddl.

NR_OF_SCP_BLOCKS has to be selected in such way that it equals the total number of SCP compatible function blocks as specified in the FF_FUNCTION_BLOCK_LIST section in the related commScript.

4.3.2 Resource block description

The resource block parameters are described in DD source files *cs_xxx_resb.ddl* and the header files *cs_xxx_resb.h*. Device-specific parameters can be defined via the macro *RESB_DEV_SPECIFIC_PARAMS* by adding all device specific parameters. If there are no device-specific parameters the macro can be empty. The added parameters have to be declared before defining variable type, access rights (Read/Write) according to the FF DD language requirements.

Example:

```
#define RESB_DEV_SPECIFIC_PARAMS                          \
          ADD HART_DEV_IDENT,            hart_dev_ident ;
```

Below you find an example of attributes of the record *hart_dev_ident*. The macros for the label and help string are defined in the header file *cs_start_hart.h*.

```
VARIABLE   hart_dev_type
{
    LABEL          LBL_HART_DEV_TYPE ;
    HELP           HLP_HART_DEV_TYPE ;
    CLASS          CONTAINED ;
    TYPE           UNSIGNED_INTEGER (2) ;
    CONSTANT_UNIT  [blank] ;
    HANDLING       READ ;
}

VARIABLE   hart_dev_rev
{
    LABEL          LBL_HART_DEV_REV ;
    HELP           HLP_HART_DEV_REV ;
    CLASS          CONTAINED ;
    TYPE           UNSIGNED_INTEGER (1) ;
    CONSTANT_UNIT  [blank] ;
    HANDLING       READ ;
}

VARIABLE   hart_sw_rev
{
    LABEL          LBL_HART_SW_REV ;
    HELP           HLP_HART_SW_REV ;
    CLASS          CONTAINED ;
    TYPE           UNSIGNED_INTEGER (1) ;
    CONSTANT_UNIT  [blank] ;
    HANDLING       READ ;
}

VARIABLE   hart_hw_rev
{
    LABEL          LBL_HART_HW_REV ;
    HELP           HLP_HART_HW_REV ;
    CLASS          CONTAINED ;
    TYPE           UNSIGNED_INTEGER (1) ;
    CONSTANT_UNIT  [blank] ;
    HANDLING       READ ;
}

VARIABLE   hart_dev_id
{
    LABEL          LBL_HART_DEV_ID ;
```

```
    HELP            HLP_HART_DEV_ID ;
    CLASS           CONTAINED ;
    TYPE            UNSIGNED_INTEGER (4) ;
    CONSTANT_UNIT   [blank] ;
    HANDLING        READ ;
}

RECORD    hart_dev_ident
{
    LABEL           LBL_HART_DEV_IDENT ;
    HELP            HLP_HART_DEV_IDENT ;
    MEMBERS
    {
        HART_DEV_TYPE,  hart_dev_type ;
        HART_DEV_REV,   hart_dev_rev ;
        HART_SW_REV,    hart_sw_rev ;
        HART_HW_REV,    hart_hw_rev ;
        HART_DEV_ID,    hart_dev_id ;
    }
}
```

Device-specific parameters have to be added to view objects. In the example below the device specific parameter *hart_dev_ident* is added to a VIEW_4 object.

Example:

```
VARIABLE_LIST   __res_2_view_1
{
    MEMBERS
    {
        __FD_PARAMS_VIEW1
    }
}
VARIABLE_LIST   __res_2_view_3
{
    MEMBERS
    {
        __FD_PARAMS_VIEW3
    }
}
VARIABLE_LIST   __res_2_view_4
{
    MEMBERS
    {
        __FD_PARAMS_VIEW4

        /* Device-specific view extension */
        ADD VL_HART_DEV_IDENT,  PARAM.HART_DEV_IDENT ;
    }
}
```

Note

View object 2 for the resource block exists but it contains only standard parameters and therefore is not contained here. If a device-specific parameter shall be visible via VIEW_2 object it is necessary to add *VARIABLE_LIST __res_2_view_2*.

4.3.3 Transducer block description

The commDevice supports a diagnostic transducer block with predefined parameters. The DD description file cs_diag_tb.ddl in Tools\DD\cs_common and the related header file cs_diag_tb.h must be included for the definition of the diagnostic transducer block.

Device-specific transducer blocks

Device-specific transducer blocks must be defined in a DD description file. The DD source files *cs_xxx_tblk.ddl* and the related header files *cs_xxx_tblk.h* are provided as examples. It is necessary to give the transducer block object the same name as used in the commScript.

As each transducer block has a predefined list of parameters Softing provides the macro *CS_STD_TRANSDUCER_PARAM* to describe these predefined parameters. Device-specific parameters have to be added after CS_STD_TRANSDUCER_PARAM.

Example:

```
BLOCK   AI_TB
{
        CHARACTERISTICS ai_tb_character ;
        LABEL           LBL_AI_TRANSDUCER_BLOCK ;
        HELP            HLP_AI_TRANSDUCER_BLOCK ;

        PARAMETERS
        {
          /* Standard parameters for transducer block*/
          CS_STD_TRANSDUCER_PARAM

          /* Device-specific transducer block parameters */
          PRIMARY_VALUE,            __primary_value ;
          PV_UNIT,                    pv_unit ;
          HART_CFG_CHANGE_CTR,        hart_cfg_change_ctr ;
          HART_TAG_DESC_DATE,         hart_tag_desc_date ;
          HART_PV_INFO,               hart_pv_info ;
          HART_DEVICE_INFO,           hart_device_info ;
          HART_ADDITIONAL_DEV_STATUS, hart_additional_dev_status ;
        }

        PARAMETER_LISTS
        {
                VIEW_1, ai_tb_view_1 ;
                VIEW_2, ai_tb_view_2 ;
                VIEW_3, ai_tb_view_3 ;
                VIEW_4, ai_tb_view_4 ;
        }

      COLLECTION_ITEMS
      {
              no_download_ai_tb,
              upload_wanted_ai_tb
      }
}
```

__primary_value … hart_additional_dev_status are the device specific parameters which are declared here.

Note

__primary_value is a standard parameter imported from the Registered DD Library of the FieldComm Group. FF standard parameters are indicated by leading "__" characters.

For the view objects of the transducer blocks the macros *CS_TBLK_VIEW_<i>*, with i = 1 … 4 are provided to define the commDevice parameters for a specific view object. These macros have to be used for every view object followed by additional device-specific parameters. For the example above with the AI_TB transducer block the view objects look as follows:

Example:

```
/*
*********************************************************************
** Variable lists
*********************************************************************
*/


VARIABLE_LIST   ai_tb_view_1
{
        LABEL           LBL_AITB_VIEW_1 ;
        HELP            HLP_AITB_VIEW_1 ;
        MEMBERS
        {
                CS_TBLK_VIEW_1

                /* Device-specific view extension */
                VL_PRIMARY_VALUE,       PARAM.PRIMARY_VALUE ;
        }
}

VARIABLE_LIST ai_tb_view_2
{
        LABEL           LBL_AITB_VIEW_2 ;
        HELP            HLP_AITB_VIEW_2 ;
        MEMBERS
        {
                CS_TBLK_VIEW_2

                /* Device-specific view extension */
                VL_PV_UNIT,                 PARAM.PV_UNIT ;
                VL_HART_CFG_CHANGE_CTR,     PARAM.HART_CFG_CHANGE_CTR ;
        }
}

VARIABLE_LIST ai_tb_view_3
{
        LABEL           LBL_AITB_VIEW_3 ;
        HELP            HLP_AITB_VIEW_3 ;

        MEMBERS
        {
                CS_TBLK_VIEW_3

                /* Device-specific view extension */
                VL_PRIMARY_VALUE,               PARAM.PRIMARY_VALUE ;
                VL_HART_ADDITIONAL_DEV_STATUS,  PARAM.HART_ADDITIONAL_DEV_STATUS ;
        }
```

```
}

VARIABLE_LIST ai_tb_view_4
{
    LABEL           LBL_AITB_VIEW_4 ;
    HELP            HLP_AITB_VIEW_4 ;
    MEMBERS
    {
                CS_TBLK_VIEW_4

                /* Device-specific view extension */
                VL_HART_TAG_DESC_DATE,         PARAM.HART_TAG_DESC_DATE ;
                VL_HART_PV_INFO,               PARAM.HART_PV_INFO ;
                VL_HART_DEVICE_INFO,           PARAM.HART_DEVICE_INFO ;
    }
}
```

4.3.4 Function block description

As the parameters of the function blocks are specified by the FieldComm Group there is no need to make device-specific adjustments. Softing provides DD files for all supported function block types:

- cs_ai.ddl Analog Input function block
- cs_ao.ddl Analog Output function block
- cs_di.ddl Discrete Input function block
- cs_do.ddl Discrete Output function block
- cs_pid.ddl PID control function block
- cs_ar.ddl Arithmetic function block
- cs_cs.ddl Control Selector function block
- cs_is.ddl Input Selector function block
- cs_sc.ddl Signal Characterizer function block
- cs_it.ddl Integrator function block

The DD files of the function blocks are in *Tools\DD\cs_common*.

The files *cs_xxx_fblks.ddl* are used to include the required DD files. A function block DD file must be included only if your device supports a specific function block type.

The *cs_xxx_fblks.ddl* files also include the DD description for the block error description parameter that is defined in *cs_block_errdsc.ddl*.

4.4 Generating DD binary files

The device description binary files are generated using an FF Tokenizer [1)] and the Registered DD library[2)].

As a commScripter user you may not have a licensed FF Tokenizer and a Registered DD Library. For generating the DD binary you can send your DD source files (*.ddl & .h*) to support.automation@softing.com. Softing will then convert the source files for you and will send the binary output files (*.ff5 and .sy5*) back to you.

Note

The costs for generating DD binaries are billed via a separately ordered support bundle.

1) The FF Tokenizer is the main component of the *DD Integrated Development Environment (DD-IDE)* offered by the FieldCommGroup.

2) The Registered DD Library is a product offered by the FieldCommGroup.

4.5 Creating PROFIBUS PA device descriptions

The device description for a PROFIBUS PA device basically contains the identification of the device and the description of blocks and block parameters. In addition a device description for a PROFIBUS PA device does contain a description of the representation of device parameters in the user interface of host systems (menus, windows, pages).

Note

In the PROFIBUS technology device descriptions are commonly named electronic device descriptions (EDDs).

The commScripter delivery package includes PROFIBUS PA device descriptions for four example devices. You can use one of the examples as a starting point for your device description.

The device descriptons provided with the commScripter package are intended to be used with SIMATIC PDM. SIMATIC PDM (Process Device Manager) is a universal, manufacturer-independent tool for configuration, parameter assignment, commissioning, diagnostics and maintenance of intelligent process devices (actors, sensors). For more information about SIMATIC PDM click here.

Main description files

The example device descriptions have the following main DD files:

- 0117_0320_01.ddl: demo device with a simple HART mapping
- 0117_0307_01.ddl: complex HART device
- 0117_0309_01.ddl: complex Modbus device

The main DD files include all files for defining function blocks, the physical block and transducer blocks.

Mandatory commDevice parameters

The physical block, the function blocks and the device-specific transducer blocks have a set of parameters which are mandatory for commDevices. The delivery package comes with a description of all these parameters, which is based on the PI EDD Import Library.

The EDD Import Library can be downloaded from the PROFIBUS & PROFINET International (PI) Organization by PI members website, software and tools download area:

https://www.profibus.com/download/edd-import-library/

Device-specific parameters

Device-specific parameters can be added to both the physical block and the device-specific transducer blocks.

4.5.1 Device identification

In order to declare the device identification parameters and device description revision the identification section in the main description files have to be modified.

```
/* ----------------------*/
/* Device Identification */
/* ----------------------*/
MANUFACTURER    279,          /* Manufacturer ID Softing 0x0117 */
DEVICE_TYPE     0x0320,       /* Sample Device Type */
DEVICE_REVISION 1,
DD_REVISION     1
```

4.5.2 Physical block description

The physical block parameters are described in EDD source file *phy_std.inc* (standard parameters) and *phy_ext.inc* (device specific parameters).

Example:

```
/* ----------------- */
/* Physical block    */
/* ----------------- */
#include "phy_std.inc"
#include "phy_ext.inc"
```

4.5.3 Function block description

As the parameters of the function blocks are specified by the PI organization, there is no need to make device-specific adjustments. Softing provides EDD source files for all supported function block types:

- funcX_AI.inc: Analog Input function block
- funcX_AO.inc: Analog Output function block
- funcX_DI.inc: Discrete Input function block
- funcX_DO.inc: Discrete Output function block

- funcX_TOT.inc: Totalizer function block

For each function block a name prefix, the number of the function block, the channel default value and the supported channel values have to be declared before the function block declaration is included.

Name prefix, function block number and channel values are declared via macros in the device specific **_macros.inc* (e.g. *0117_0320_01_macros.h*) files and the *prefix.h file*.

```
/* ----------------- */
/* AI function block */
/* ----------------- */
#define FBNR 1
#define PF_AI(NAME)        PF_AI1(NAME)
#define AI_X_CHANNEL_DFT   AI_1_CHANNEL_DFT
#define AI_X_CHANNEL_LIST  AI_1_CHANNEL_LIST
#define LBL_AI_X           LBL_AI_1
#include "funcX_AI.inc"
#undef LBL_AI_X
#undef AI_X_CHANNEL_DFT
#undef AI_X_CHANNEL_LIST
#undef PF_AI
#undef FBNR
```

4.5.4 Transducer block description

Device-specific transducer blocks

The device-specific transducer blocks must be defined in an EDD description file. The EDD source file *transX_std.inc* contains the description of the standard parameters. As with the function block description source files a name prefix and a transducer block number must be defined using the macros TBNR and PF_TRANS.

Example:

```
#define TBNR 1
#define PF_TRANS(NAME) PF_TRANS1(NAME)
#include "transX_std.inc"
#undef PF_TRANS
#undef TBNR

#include "trans1_ext.inc"
```

The device-specific transducer block parameters have to be described with an include file e. g. *trans1_ext.inc*:

Example:

```
/*****************************************************************************/
/************************** VARIABLEs ****************************************/
/*****************************************************************************/
VARIABLE  trans1_ext_PRIMARY_VALUE_value
{
  LABEL     [PI_PRIMARY_VALUE_Value_label];
  HELP      [PI_empty];
  CLASS     CONTAINED & DYNAMIC;
  TYPE      FLOAT;
  HANDLING  READ;
}
```

```
/*************************** COMMANDs ************************************/
COMMAND trans1_ext_PRIMARY_VALUE_read
{
  BLOCK trans1_std_block;
  INDEX 8;
  OPERATION READ;
  RESPONSE_CODES DPV1_PA_rsp_codes;
  TRANSACTION
  {
    REQUEST
    {
    }
    REPLY
    {
      trans1_ext_PRIMARY_VALUE_value,
      trans1_ext_PRIMARY_VALUE_status
    }
  }
}
```

Diagnostic transducer block

CommDevices support a diagnostic transducer block with predefined parameters. The EDD description file *transX_diag.inc* must be included in the definition of the diagnostic transducer block for the device specific parameters.

Example:

```
/* ---------------------------------- */
/* Softing diagnosis transducer block */
/* ---------------------------------- */
#define TBNR 2
#define PF_TRANS(NAME) PF_TRANS2(NAME)
#include "transX_std.inc"
#include "transX_diag.inc"
#undef PF_TRANS
#undef TBNR
```

4.5.5 View objects

View objects are used for an optimized read access to block parameters. The EDD defines commands in order to read the block parameters which are contained in view objects as a block of data instead of defining commands to read these parameters one by one.

For the physical block the command declaration to read parameters contained in the view 2 object is defined in *phy_std.inc*. For details about the physical block views see Appendix 6.10.1 145

For the AI function block the command declaration to read parameters contained in the view 2 object is defined in *funcX_AI.inc*. It reads the standard parameters ST_REV, MODE_BLK, ALARM_SUM, OUT and SIMULATE.

For the DI function block the command declaration to read parameters contained in the view 2 object is defined in *funcX_DI.inc*. It reads the standard parameters ST_REV, MODE_BLK, ALARM_SUM, OUT_D and SIMULATE.

For the AO function block the command declaration to read parameters contained in the view 2 object is defined in *funcX_AO.inc*. It reads the standard parameters ST_REV, MODE_BLK, ALARM_SUM, SP, READBACK, RCAS_IN, RCAS_OUT, POS_D, SETP_DEVIATION, CHECK_BACK, CHECK_BACK_MASK, SIMULATE and OUT.

For the DO function block the command declaration to read parameters contained in the view 2 object is defined in *funcX_DO.inc*. It reads the standard parameters ST_REV, MODE_BLK, ALARM_SUM, SP_D, OUT_D, READBACK_D, RCAS_IN_D, RCAS_OUT_D, SIMULATE, CHECK_BACK, CHECK_BACK_MASK.

As for transducer blocks the view objects are implemented device specifically, the definition of the corresponding COMMAND declarations is done in the transducer block include files.

Example:

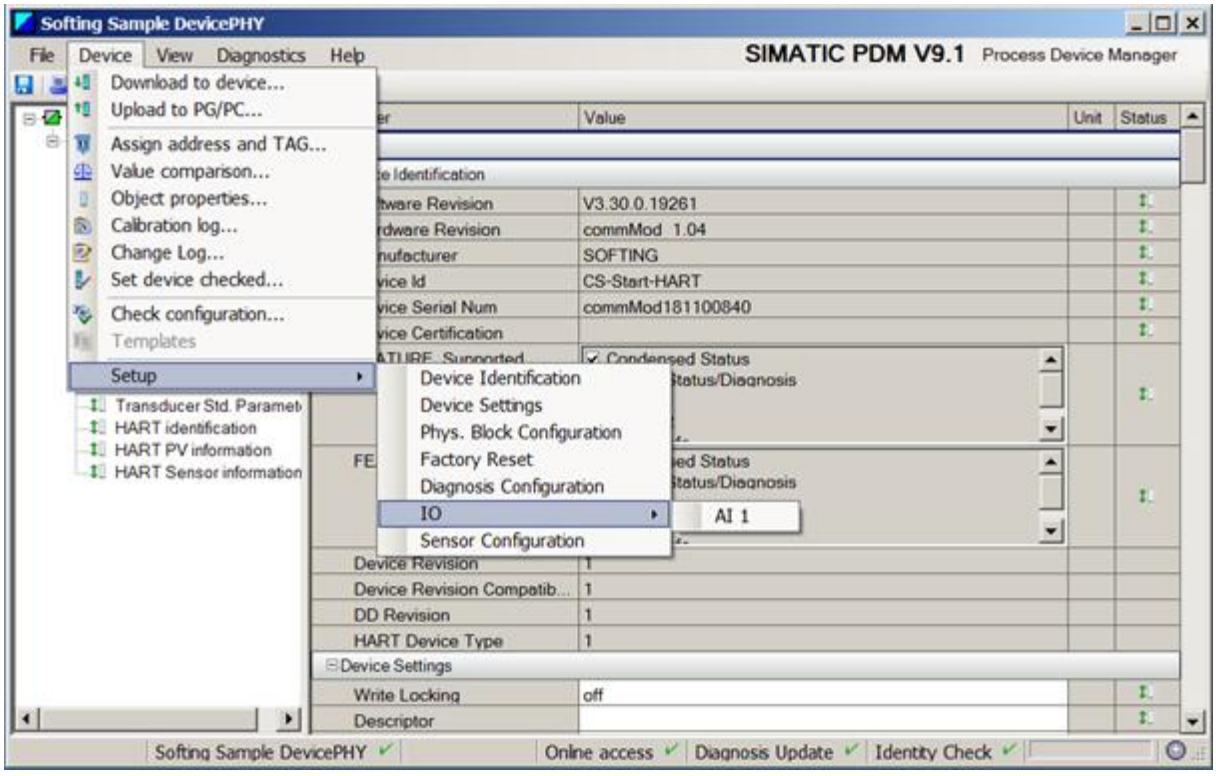
```
/*************************** COMMANDs for VIEW objects ************************/
COMMAND trans1_ext_VARLIST_2_read
{
  BLOCK trans1_std_block;
  INDEX 215;
  OPERATION READ;
  RESPONSE_CODES DPV1_PA_rsp_codes;
  TRANSACTION
  {
    REQUEST
    {
    }
    REPLY
    {
    trans1_std_ST_REV,
    trans1_std_MODE_BLK_Actual,
    trans1_std_MODE_BLK_Permitted,
    trans1_std_MODE_BLK_Normal,
    trans1_std_ALARM_SUM_Current,
    trans1_std_ALARM_SUM_Unacknowledged,
    trans1_std_ALARM_SUM_Unreported,
    trans1_std_ALARM_SUM_Disabled,
    trans1_ext_PV_UNIT,
    trans1_ext_HART_CFG_CHANGE_CTR
    }
  }
}
```

4.5.6 Menus

An EDD must contain statements that describe the graphical representation of device parameters in an EDD application. The EDD source code in the commScripter delivery package contains descriptions for the graphical representation of physical block parameters, function block parameters and transducer block parameters, and a hierarchy of menus for browsing between the graphical user interface elements.

The parameters are grouped according to their intended usage and displayed in tabbed windows. As a starting point for browsing the top level menus "Device", "Diagnostics", "Process Variables" and "Offline" are defined in the **_menus.inc* include files (e.g. *0117_0320_01_menus.h*).

If additional device specific parameters are implemented for the physical block or transducer blocks, the tabbed windows representing these parameters have to be defined in the *phy_ext.inc* and the *trans*_ext.inc* files.



4.5.7 Upload/download parameter lists

For offline parameterization (configuring a device without communicating with the physical device) an upload data set and a download data set has to be defined in the EDD.

The upload parameters are defined using the MENU download_variables, the download parameters are defined in the MENU upload_variables entries in the **_menus.inc* include files (e.g. *0117_0320_01_menus.inc*).

The download parameter list contains all writable physical block, function block and transducer block parameters except the FACTORY_RESET physical block parameter. Additional device specific parameters must be added to the list.

The upload parameter list contains all parameters contained in the download parameter list and additionally some important read only device parameters, which are referenced in the offline graphical parameter representation.

4.5.8 Standard dictionary

Commonly used texts for the graphical representation of device parameters by the EDD application (names of parameters, enumeration list texts, help texts and unit code textual representations) are contained in a dictionary file. The commScripter delivery package provides the dictionary file *standard.dct*, which is based on the PI EDD Import Library.

References to the standard dictionary texts use the identifier in square brackets:

Example:

```
VARIABLE phys_ST_REV
{
    LABEL [PI_ST_REV_label];
    HELP [PI_ST_REV_help];
    CLASS CONTAINED;
    HANDLING READ;
    TYPE UNSIGNED_INTEGER(2)
    {
           DEFAULT_VALUE 0;
    }
}
```

4.5.9 Integration into SIMATIC PDM

Device descriptions are integrated into SIMATIC PDM using the Device Integration Manager, which is a part of SIMATIC PDM. The syntax and semantics of the device description are checked during the integration procedure.

A SIMATIC PDM devices file supplies catalog information for the integration into SIMATIC PDM. The file *Softing_Sample_0320.devices* is an example for a device, which appears in the SIMATIC PDM device catalog as a PROFIBUS PA device in a subdirectory - Sensors - Level - Echo - SOFTING AG - Softing CS.

Example:

```
[Softing_Sample]
Name          = "Softing CS"
Folder        = "Softing"
Description   = "Softing Sample commScript Device"
Manufacturer  = "Softing"
Communication = "PA"
Class         = "Sensor"
Subclass      = "Level"
Subclass2     = "Echo"
GSD           = "SOFT110E.gsd"
DDL           = "0117_0320_01.ddl"
DicFile       = "standard.dct"
```

4.6 Generating a commScript configuration table

The commScript configuration table is generated by the commScripter tool. The commScripter processes commScripts which define blocks, parameters and other attributes of a commDevice. A commScript is a text file containing a list of sections. Each section defines a specific aspect of a commDevice function.

- Each section starts with <Section_name> and ends with END_<Section_name>
- There are mandatory and optional sections.
- Some sections are common for all field devices other sections describe the mapping of HART or Modbus parameters to FF/PA block parameters.

Common device sections

(valid for all devices, FF+HART, FF+MODBUS, PA+HART, PA+MODBUS)

Section name	Requirement
DEVICE_SETTINGS	Mandatory
CHANNEL_LIST	Mandatory
UNIT_LIST	Conditional *
DATA_STRUCT_LIST	Optional

* UNIT_LIST is mandatory if AI or AO blocks are used.

FF device sections

(valid for all types of FF devices)

Section name	Requirement
FF_DEVICE_IDENT	Mandatory
FF_DEVICE_PARAM	Mandatory
FF_FUNCTION_BLOCK_LIST	Mandatory
FF_SCP_SETTINGS	Optional
FF_RESOURCE_BLOCK	Optional
FF_TRANSDUCER_BLOCK	Mandatory
FF_FDIAG_LIST	Optional
FF_EXT_DIAG_LIST	Optional
FF_RESB_ERROR_LIST	Optional
FF_TB_ERROR_LIST	Optional
FF_TB_XD_ERROR_LIST	Optional
FF_FBLK_DEFAULT_VALUE_LIST	Optional
FF_SCP_FBLK_DEFAULT_VALUE_LIST	Mandatory

PA device sections
(valid for all types of PA devices)

Section name	Requirement
PA_DEVICE_IDENT	Mandatory
PA_FUNCTION_BLOCK_LIST	Mandatory
PA_TRANSDUCER_BLOCK	Mandatory
PA_PHYSICAL_BLOCK	Optional
PA_DIAG_LIST	Optional
PA_ADD_DIAG_LIST	Optional
PA_FBLK_DEFAULT_VALUE_LIST	Mandatory

HART sections

Section name	Required
HART_COMM	Optional
HART_CMD_MAPPING	Mandatory
HART_DIAG_MAPPING	Optional
HART_VENDOR_UNIT_MAP	Optional

Modbus sections

Section name	Required
MODB_COMM	Mandatory
MODB_PARAM_MAP	Mandatory
MODB_STD_UNIT_MAP	Optional
MODB_VENDOR_UNIT_MAP	Optional
MODB_FACTORY_RESET	Optional
MODB_DIAG_MAP	Optional

4.6.1 General device settings

4.6.1.1 Script revision management

commScript configuration tables contain 2 levels of revision information: the "ScriptRev", which indicates the internal structure of the configuration table, and the "ConfigRev" denoting user defined version information about the script contents.

"ScriptRev" is fixed by the commScripter version which generated the configuration table. If it does not match the structure as expected by the commModule firmware, the mapping cannot be used at all. In this case, as no user defined identification information is available, the device will start with a default identification clearly indicating the mismatch between commScripter and firmware version. Even if normal device operation is not possible here, the device will contain a resource block or physical block with a 'CS_SCRIPT_REV' parameter of type USIGN16 to read out the expected (by the firmware) script revision in the high byte and the actual revision (as written by commScripter into the configuration table) in the low byte.

"ConfigRev" can be used by the script developer for version management of the mapping. No version check is done here by the firmware, its value is just provided within the 'CS_CONTENT_REV' parameter of the resource block / physical block.

FF configuration table mismatch

If during startup the FF commDevice detects a configuration table revision which it is not able to cooperate with, or no valid configuration table at all, it enters network communication in SM state INITIALIZED with a predefined PD_TAG and a predefined DEV_ID.

The commDevice reject any SET_PD_TAG and SET_ADDR request so that it will not be possible to change to a different SM state.

In this state it is not possible to establish connections to the MIB or FBAP VFD of the device. Therefore, it is not necessary to define which FBAP blocks are available when this conflict situation occurs because they cannot be accessed.

If a table revision mismatch is detected the commDevice starts with:

- PD-TAG: CS_ERR_VER_XX_VERSUS_VER_YY
- DEV-*ID: 1E6D111111111111111111*

Where XX is the supported revision and YY is the revision of the downloaded configuration table.

If an invalid configuration table is detected the commDevice starts with

- PD-TAG: CS_ ERR_INVAL_MAPPING_TABLE
- DEV-*ID: 1E6D111111111111111111*

PA configuration table mismatch

If during startup the PA commDevice detects a configuration table revision which it is not able to cooperate with, or no valid configuration table at all, it enters network communication with special device identification and attributes:

- Slave Address 126

- Ident-Nr: 0x9770 (Which is reserved by PI but currently not used)
- Device provides a physical block on Slot 0 but no function or transducer blocks
- PB.DEVICE_ID: "SoftingCS_ERR"
- EXT_DIAG set in Slave Diagnosis Status_1

If a table revision mismatch is detected the commDevice indicates this by

- *I&M0 Order ID: "CS_ERR_MISMATCH "*
- *PB.DEVICE_CERTIFICATION: "CS_ERR_MISMATCH_XXYY"*

Where XX is the supported revision and YY is the revision of the downloaded mapping table.

If an invalid configuration table is detected the commDevice starts with

- I&M0 Order ID: "CS_ERR_CFG_INVALID "
- PB.DEVICE_CERTIFICATION: "CS_ERR_CFG_INVALID"

4.6.1.2 Section DEVICE_SETTINGS

It is necessary to specify which current is consumed by the commModule MBP. The required current must be defined with the keyword PowerRequired in the mA range from 10 mA to 26 mA . You can also enable the 3V_OUT and/or the 6V_OUT power output lines (see the commModule MBP Hardware Manual).

Parameter description

Item	Description
PowerRequired	Quiescent current (in milliamps) drawn from the fieldbus. Unit: mA Type: unsigned integer Valid range: 10 to 26
3V_OUT	Specifies if commModule MBP supplies 3V at the 3V_OUT pin. Type: BOOL Valid range: True, False Optional. Preset value: True
6V_OUT	Specifies if commModules supplies 6V at the 6V_OUT pin. Type: BOOL Valid range: True, False Optional. Preset value: False
StartDelay	Specifies the time which shall be waited after powering the external device via 3V_OUT or 6V_OUT before sending the first serial Modbus or HART frame. Type: unsigned integer Unit: ms Valid range: 0 to 10000 Optional: Preset value: 0 [ms]
ConfigRev	Revision of the commScript configuration table. You may use the ConfigRev to identify different revisions of your commScript configuration table. The revision of the commScript configuration table is shown in the Resource Block/Physical Block parameter CS_CONTENT_REV.

Item	Description
	Type: USIGN16 Valid range: 1 to 65535 Optional. Preset value: 1
CheckConfigRev	Indicates if the value of ConfigRev given in the commScript table should be compared to the previously used during startup. If the ConfigRev value was changed the factory default settings would be restored. Type: BOOLEAN Values: False: Don't compare current and last ConfigRev True: Compare ConfigRev; use factory defaults if they are different Optional. Preset value: False
SwRevision	Software revision of the device overriding revision set by commScripter. This value is displayed in the resource block SOFTWARE_REV parameter or physical block SOFTWARE_REVISION parameter if no HART/MODBUS mapping is provided in the mapping sections. Type: Visible string Valid range: 1 to 32 octets Optional. Preset value: software revision of commModule.
HwRevision	Hardware revision of the device overriding revision set by commScripter. This value is displayed in the resource block HARDWARE_REV parameter or physical block HARDWARE_REVISION parameter if no HART/MODBUS mapping is provided in the mapping sections. Type: Visible string Valid range: 1 to 32 octets Optional. Preset value: hardware revision of commModule.
HwWriteProtect	Defines a GPIO pin of the commModule used as HW write protect pin according to FF or PA V3.02 specification. Type: String Values: "Pin<GPIOPinNr>" or "Pin <GPIOPinNr" -> List of possible GPIO pins is below Optional: If this keyword is not defined, HW write protection is not supported
HwWriteProtectLevel	The keyword defines the active level of the GPIO pin used for HW protection Type: String Values: "Low": active level is low/0. "High: active level is high/3.15 V Optional. Default value is "Low" If the keyword is defined but HwWriteProtect is not defined, this entry is ignored.
LedActivity	Defines a GPIO pin of the commModule used for activity LED. In the FF field device firmware the activity LED is used to display if the device is in the token ring i.e. actively communicating. The LED will be switched on when the device is in the token ring and switched off when the device is not in the token ring. For the PA firmware this LED gives an indication that the firmware is executing correctly. The LED is toggled when normal operation of the firmware is performed. Type: String Values: "Pin<GPIOPinNr>" or "Pin <GPIOPinNr" -> List of possible GPIO pins is below

Item	Description
	Optional: If this keyword is not defined, the activity LED will not be used in the firmware
LedActivityLevel	The keyword defines the active level of the GPIO pin used to control the activity LED i.e. the level where the LED will be switched on. Type: String Values: “Low”: active level is low/0 - “High: active level is high/3.15 V Optional. Default value is “Low” If the keyword is defined but LedActivity is not defined, this entry is ignored.
LedDebug	Defines a GPIO pin of the commModule used for debug LED. In the debug version of the FF field device firmware the LED gives an indication that the firmware is executing correctly. The LED is toggled when normal operation of the firmware is performed For the PA firmware and for the release version of the FF firmware the LED is not used. Type: String Values: “Pin<GPIOPinNr>” or “Pin <GPIOPinNr” -> List of possible GPIO pins is below Optional: If this keyword is not defined, the debug LED will not be used in the firmware.
NvramErase	Defines a GPIO pin of the commModule used as NVRAM erase pin. This pin will be checked during startup. If the NVRAM erase pin is set to active level at this time the configuration data of the commModule will be deleted and factory settings will be used. During operation the pin is not checked anymore. Type: String Values: “Pin<GPIOPinNr>” or “Pin <GPIOPinNr” -> List of possible GPIO pins is below Optional: If this keyword is not defined, NVRAM configuration will not be cleared on startup.
NvramEraseLevel	The keyword defines the active level of the GPIO pin used for NVRAM erase pin Type: String Values: “Low”: active level is low/0. “High: active level is high/3.15 V Optional. Default value is “Low” If the keyword is defined but NvramErase is not defined, this entry is ignored.

The following list shows the GPIO pins that can be used for HwWriteProtect, NvramErase, LedActivity, LedDebug.

GPIO pin number	commModule pin description
4	GPIO_PA4
5	GPIO_PE6 (Note pin can also be used as AD input)
9	GPIO_PE2
10	GPIO_PE5
11	GPIO_PE7

12	GPIO_PE1
23	GPIO_P23
24	GPIO_P22
25	GPIO_P21
26	GPIO_P20
27	GPIO_PC6
29	GPIO_PC5
30	GPIO_PC4
33	GPIO_PD2

Syntax:

<Item> = <Value>

Example:

```
DEVICE_SETTINGS
      PowerRequired       = 18
      3V_OUT              = True
      6V_OUT              = False
      ConfigRev           = 1
      CheckConfigRev      = False
      SwRevision          = "FD-SW V3.60.0 StartHART"
      HwRevision          = "cM V1.23"
      HwWriteProtect      = "Pin33"       // Port D.2
      LedActivity         = "Pin12"       // Port E.1
      LedDebug            = "Pin9"        // Port E.2
      NvramErase          = "Pin26"       // Port 2.0
      HwWriteProtectLevel = "Low"
      LedActivityLevel    = "Low"
      LedDebugLevel       = "High"
      NvramEraseLevel     = "Low"
END_DEVICE_SETTINGS
```

4.6.1.3 FF devices

Section FF_DEVICE_IDENT

An FF device is identified via manufacturer id, device type and device revision. The manufacturer id is a 6 byte value which is determined by the FieldComm Group. The type of the device is determined by the device manufacturer via a two byte device type.

Parameter description

Item	Description
ManufacturerId	Manufacturer identification number as determined by the FieldComm Group Type: unsigned integer Valid range: 0x000100 - 0x7FFFFF
DeviceType	Manufacturer's model number associated with the device Type: unsigned integer Valid range: 1 to $2^{16} - 1$
DeviceRevision	Manufacturer's revision number associated with the device Type: unsigned integer Valid range: 1 to $2^{8} - 1$
DDRevision	Specifies the oldest revision number (numerically smallest) of DD, which describes the devices of this device revision. Type: unsigned integer Valid range: 1 to $2^{8} - 1$ Optional; preset value: 1
FbapVfdTag	Name of the "Function Block Application Process Virtual Field Device" (FBAP-VFD). (The FBAP-VFD is an object list containing all block parameters of a device.) The FbapVfdTag appears in the VFD reference list of the system management. Visible string: 4 to 32 octets, no leading underscores The string must be encapsulated by quote marks Optional; preset value: "FBAP-VFD"
VendorName	Name of the FBAP-VFD vendor. Is part of the FMS Identify response data. Visible string: 2 to 32 octets The string must be encapsulated by quote marks.
ModelName	Model name of the FBAP-VFD. Is part of the FMS Identify response data Visible string: 2 to 32 octets The string must be encapsulated by quote marks
Revision	Revision of the FBAP-VFD. Is part of the FMS Identify response data Visible string: 2 to 32 octets The string must be encapsulated by quote marks.

Syntax:

<Item> = <Value>

Example:

```
FF_DEVICE_IDENT
  ManufacturerId  = 0x1E6D11
  DeviceType      = 0x0320
  DeviceRevision  = 6
  VendorName      = "Softing"
  ModelName       = "CS-Start-HART"
  Revision        = "3.60"
END_FF_DEVICE_IDENT
```

Section FF_DEVICE_PARAM

Parameter description

Item	Description
Default_PD_Tag	Visible string: 4 to 16 octets, no leading underscores The string must be encapsulated by quote marks
MinCycleTime	Time duration of the shortest macrocycle of which the device is capable Unit: 1/32 millisec Type: unsigned integer Valid range: 3200 to 64000 (0.1 to 2.0 sec) Optional; preset value: 8000 (0.25 sec)

Syntax:

<Item> = <Value>

Example:

```
FF_DEVICE_PARAM
  Default_PD_Tag = "CS Start HART"
END_FF_DEVICE_PARAM
```

Section FF_SCP_SETTINGS

This section allows the device manufacturer to customize the initial configuration of the SCP Transducer block.

Parameter description

Item	Description
ScpActive	Specifies if the field device should start in SCP mode in its initial state. This permits devices to be shipped with a factory default (out-of-the-box) with SCP mode enabled. Type: BOOL Valid range: True, False Preset value: True

Syntax:

<Item> = <Value>

Example:

```
FF_SCP_SETTINGS
  ScpActive = False
END_FF_SCP_SETTINGS
```

4.6.1.4 PA devices

Section PA_DEVICE_IDENT

Each PROFIBUS PA commDevice has to support a profile-specific ident number and a manufacturer-specific ident number:

- Profile-specific ident numbers are issued by the PROFIBUS and PROFINET International (PI) organization. The supported profile-specific ident numbers are listed in the Appendix 143.
- Manufacturer-specific ident numbers are assigned by the PI. You can request a device ident number at https://www.profibus.com/products/ident-numbers.

In addition each PROFIBUS PA device requires a manufacturer ident number. Manufacturer IDs are assigned by the PI. You can request a manufacturer ID at https://www.profibus.com/products/im-support.

Parameter description

Item	Description
ProfileIdentNumber	Profile specific ident number Type: unsigned integer
ManufacIdentNumber	Manufacturer specific ident number as assigned by the PI Type: unsigned integer Valid range: 0x0001 to 0xFFFE
DeviceManufacId	Manufacturer identification code as assigned by the PI Type: unsigned integer Valid range: 0 … 65535
DeviceId	Manufacturer-specific device identification string Type: visible string Valid range: 2 to 16 octets The string must be enclosed within quotation marks.
VendorName	The vendor name appears in the GSD file. Type: visible string Valid range: 2 to 32 octets The string must be enclosed within quotation marks.
ModelName	The model name appears in the GSD file. Type: visible string Valid range: 2 to 32 octets The string must be enclosed within quotation marks.
OrderId (optional)	Identification & Maintenance order number. Type: visible string Valid range: 0 to 20 octets The string must be enclosed within quotation marks. The default value "No Order ID" is used.
DeviceRevision (optional)	Manufacturer's revision number associated with the device Type: unsigned integer

Item	Description
	Valid range: 1 to 2^{16} -1 In case this optional parameter is not declared in the script the default value 1 is used.
DeviceRevisionComp (optional)	Represents the lowest device revision which is supported by the device. Type: unsigned integer Valid range 1 to 2^{16} -1 In case this optional parameter is not declared in the script, the default value 1 is used.
DDRevision (optional)	Reserved for future use. Type: unsigned integer Valid range: 1 to 2^{16} -1 In case this optional parameter is not declared in the script, the default value 1 is used.
DeviceCertification (optional)	Value for Physical BLock parameter DEVICE_CERTIFICATION. Type: visible string Valid range: 0 to 32 octets The string must be enclosed within quotation marks. As default a visible string with 32 blanks is used.

Syntax:

<Item> = <Value>

Example:

```
PA_DEVICE_IDENT
  ProfileIdentNumber = 0x9700         // 1-AI Transmitter
  ManufacIdentNumber = 0x110E         // Softing Ident Number commModule MBP
  DeviceManufacId    = 279            // Softing
  DeviceId           = "CS-Start-HART"
  VendorName         = "Softing"
  ModelName          = "commModule MBP"
  OrderId            = "MyOrderID"
  DeviceRevision     = 1
  DeviceRevisionComp = 1
  DDRevision         = 1

END_PA_DEVICE_IDENT
```

4.6.2 Function blocks

4.6.2.1 Section [FF|PA]_FUNCTION_BLOCK_LIST

The FF/PA function blocks are implemented in the commModule. It is not necessary to describe function block parameters in the commScript. You only have to consider which and how many function blocks are necessary for your device. For example if your HART or Modbus device provides four temperature values the FF/PA device should support four Analog Input blocks.

For each supported block type the number block per type has is specified. The list of supported block type includes:

- NumberOfAiBlocks
- NumberOfAoBlocks
- NumberOfDiBlocks
- NumberOfDoBlocks
- NumberOfPidBlocks (FF only)
- NumberOfArBlocks (FF only)
- NumberOfScBlocks (FF only)
- NumberOfIsBlocks (FF only)
- NumberOfCsBlocks (FF only)
- NumberOfItBlocks (FF only)
- NumberOfTotBlocks (PA only)

All *NumberOfXxBlocks* are optional. The default value is 0. A commDevice must support at least one I/O function block of type AI, AO, DI or DO. The maximum number of function blocks in a commDevice is 20.

Syntax:

<Item> = <Value>

Example:

```
[FF|PA]_FUNCTION_BLOCK_LIST
    NumberOfAiBlocks = 4
END_[FF|PA]_FUNCTION_BLOCK_LIST
```

4.6.2.2 Section FF FUNCTION BLOCK LIST

Additionally for all FF devices, I/O blocks which are affected by the SCP mode have to be specified in this section. This can be done by using one of the following keywords:

- NumberOfScpAiBlocks (No. <= NumberOfAiBlocks)
- NumberOfScpAoBlocks (No.<= NumberOfAoBlocks)
- NumberOfScpDiBlocks (No.<= NumberOfDiBlocks)
- NumberOfScpDoBlocks (No.<= NumberOfDoBlocks)

The total number of SCP blocks must be 1 <= n <= 8 and for each individual block type the number has to be lower or equal than the available number of blocks from the same type. At least 1 I/O function block must be declared as SCP compatible.

Example:

```
FF_FUNCTION_BLOCK_LIST
    NumberOfAiBlocks = 4
    NumberOfScpAiBlocks = 2
    NumberOfDiBlocks = 2
END_FF_FUNCTION_BLOCK_LIST
```

4.6.3 Parameters, data types and data structures

4.6.3.1 Parameters

There are three parameter classes: simple variables, records and arrays. The length of read-only commDevice parameters is restricted to 122 octets. Writeable parameters have a maximum length of 119 octets.

4.6.3.2 Data types

The following data types are supported. The valid range for the length (len) of octet strings and visible strings is 2 to 122/119, where 122 is reserved for *read only* and 119 for writeable parameters.

- UNSIGNED8
- UNSIGNED16
- UNSIGNED32
- INTEGER8
- INTEGER16
- INTEGER32
- FLOAT
- OCTET_STRING(len)
- VISIBLE_STRING(len)*
- BIT_STRING(1)
- BIT_STRING(2)
- BIT_STRING(4)
- DATE**

* Softing recommends a maximum of 32 bytes for visible strings.

** Data type DATE can be used for unmapped parameters only (see also Sections 4.6.4.1 55 and 4.6.5.1 61).

4.6.3.3 Section DATA_STRUCT_LIST

Device specific data structures are specified by a structure name and a list of structure elements.

- A maximum number of 40 device-specific data structures is supported
- Each structure must have at least 2 elements
- The maximum number of elements per data structure is 38

Parameter description

Item	Description
StructName	Name of the data structure. Struct name length: 4 to 32 octets. No leading underscores. Allowed characters: 'A'-'Z', 'a'-'z', '0'-'9', '_' The string must not be encapsulated by quote marks The data structure name is used within the script to specify the attributes of device-specific resource, physical and transducer block parameters. (See chapter 6, Device-specific parameters).The name is used to identify the structure in the commScript. It does not appear in any PA commDevice parameter.
ElementType	Data type of the element (see Data types (53)).

Syntax

<Item> = <Value>

Example

```
DATA_STRUCT_LIST
  StructName      = HART_CMD_0_STRUCT
  ElementType     = UNSIGNED16        // HART Device Type
  ElementType     = UNSIGNED16        // HART Device Type
  ElementType     = UNSIGNED8         // HART Device Revision
  ElementType     = UNSIGNED8         // HART SW Revision
  ElementType     = UNSIGNED8         // HART HW Revision
  ElementType     = UNSIGNED32        // HART Device ID
DATA_STRUCT_LIST
```

4.6.3.4 Standard data structures

In the FF specification Function block (Part 1), FF-890, a list of standard data structures is defined. Three of these standard data structures can be used in commScripts:

- DS_FLOAT_S (Data struct index 65)
- DS_DISC_S (Data struct index 66)
- DS_SCALE (Data struct index 68)

It is not necessary to define DS_FLOAT_S, DS_DISC_S and DS_SCALE in the commScript. For the elements of this records see Standard FF data structure (135).

The PROFIBUS PA V3.02 specification defines similar standard data structures: DS-101, DS-102 and DS-36. For PA devices the commScripter maps DS_FLOAT_S, DS_DISC_S and DS_SCALE to the appropriate PA data structures. Therefore the standard FF data structures can also be used to define PA block parameters.

4.6.4 Resource block and FF transducer blocks

4.6.4.1 Device-specific parameters

Parameter description

Item	Description
Parameter class	STANDARD_PARAM: The parameter is defined by an FF specification PARAM: A device-specific parameter
Parameter name	Visible string: 4 to 32 octets, no leading underscores. Parameter name must be identical to the name used in the device description.
Parameter flags (for commModule connected with HART or Modbus device)	W: Parameter is writeable. W_OOS: Parameter is writeable in mode OOS only. W_MAN: Parameter is writable in modes MAN and OOS. U: Parameter is not mapped to HART read/write commands or to Modbus registers. X: Parameter is writeable, but only the current value can be written. No other value is permitted. Has to be combined with 'U' flag S: Static Parameter. Parameter is stored non-volatile, and the static revision counter (ST_REV) is incremented when the parameter value changes N: Parameter is stored non-volatile, but the static revision counter will not be incremented when the parameter value changes. D: Parameter is dynamic. It is not stored non-volatile. C: Constant parameter (read-only). **Supported only up to version 3.50** CMD Command parameter. This is supported only if HART mapping is selected. The parameter will be mapped to a HART write command, but no HART read mapping is required. Before written via the fieldbus the parameter value is set to 0. For Modbus writeable parameters are mapped to Holding registers. Here it is always possible to read them, so this parameter flag is not supported for MODBUS mapping. SYN Synchronous access parameter. Usually, parameter values of parameters that are mapped to HART or Modbus will be accessed cyclic via HART or Modbus in the background. The cyclic period depends on the number of parameters that are accessed, the size of the parameters and the baud rate that is used for the HART or Modbus protocol. When a read access via the fieldbus is done the read returns the parameter value from the last cycle. A SYN parameter, however, will not be read in the background but directly via HART or Modbus when the parameter is read via the fieldbus. Thus, the latest available parameter from the HART/Modbus device is retrieved but the FF/PA read response will be delayed due to the serial communication via HART/Modbus. Therefore, it is not recommended to use SYN parameters in views. For Modbus the parameter data should reside in a continuous register range so that it is possible to retrieve the data with one Modbus read request.
Object class	FMS object class: Simple, Record, Array In case of arrays the number of elements is part of the object class name, e.g. Array(5) - for an array with five elements.
Parameter type	<Data type name>|<Data struct name> In case of octet and visible strings the string length is part of the data type name, e.g. VISIBLE_STRING(16).

Item	Description
Default values and constant values	See Default and constant values 79 for accepted default value format. Array and record elements are separated by white spaces. If no default values are specified, data type specific initial values 55 are used. See table with initial values in Chapter Default values and constant values 79.

Syntax:

PARAM <Parameter name> <Parameter flags> <Object class> <Parameter type> [optional] <default/constant values>

Example:

```
STANDARD_PARAM PRIMARY_VALUE                        Record     DS_FLOAT_S
PARAM          PV_UNIT                        S W_OOS Simple     UNSIGNED16
PARAM          HART_ADDITIONAL_DEV_STATUS           Array(13)  BIT_STRING(1)
```

Example of default/const values

```
PARAM UNMAP_PAR_OSTRING3     U S W    Simple    OCTET_STRING(3)   10,11,12
PARAM TB_UNMAP_VSTRING8_AR   U S W    Array(2)  VISIBLE_STRING(3) "un1" "un2"
PARAM TB_UNMAP_RECORD        U S W    Record    HART_STRUCT       "Tag" 1 2 3
```

4.6.4.2 Section FF_RESOURCE_BLOCK

Per default the commDevice resource block has 72 parameters. Device-specific parameters can be added. Only the device-specific parameters are listed in the commScript.

View lists with device-specific parameter follow the parameter list. The standard view parameters required by the FF specification are added automatically and don't have to be specified here. For details on the resource block views see Appendix 138.

Parameter description

Item	Description
Default block tag	Block tag length: 4 to 16 octets. No leading underscores. Allowed characters: 'A'-'Z', 'a'-'z', '0'-'9', '_' The default block tag of resource block is optional. If no block tag is specified the standard tag “RESOURCE” is used. The default block tag combined with the serial number of the commModule MBP appears in the block header of the resource block.

Syntax:

```
FF_RESOURCE_BLOCK  [optional] <Default block tag>
    <List of additional parameters>
        VIEW_1
         <List of parameter names>
        END_VIEW_1
        VIEW_2
         <List of parameter names>
        END_VIEW_2
        VIEW_3
         <List of parameter names>
        END_VIEW_3
        VIEW_3
         <List of parameter names>
        END_VIEW_3
        VIEW_4
         <List of parameter names>
        END_VIEW_4
        VIEW_4
         <List of parameter names>
        END_VIEW_4
        VIEW_4
         <List of parameter names>
        END_VIEW_4
END_FF_RESOURCE_BLOCK
```

Example:

```
FF_RESOURCE_BLOCK HART_RES
PARAM       HART_DEV_IDENT                  Record    HART_CMD_0_STRUCT
PARAM       HART_MESSAGE              S W   Simple    VISIBLE_STRING(32)
PARAM       ACME_DEV_CHARACTERISTICS        Record    ACME_DEV_CHARACT_STRUCT
VIEW_4
  HART_MESSAGE
END_VIEW_4
END_FF_RESOURCE_BLOCK
```

4.6.4.2.1 View objects

FF defines four view objects for each block:

- View 1 - access to dynamic operation parameter values.
- View 2 - access to static operation parameter values.
- View 3 - access to access all dynamic parameter values.
- View 4 - access to other static parameter values.

The first parameter of each view object is the *ST_REV* parameter.

The maximum length of one view object is 122 bytes. If the parameters of a complex block do not fit into a single view object it is necessary to define multiple VIEW_3 and VIEW_4 objects. (Number of VIEW_3 and Number of VIEW_4 are attributes of each block header.)

A view object must not have more than 32 elements. Even if the length of the view object is less than 122 bytes (e.g. the view contains 32 UNSIGND16 values) the view has to be split into multiple objects.

Multiple VIEW_3 and VIEW_4 objects

The second, third and further VIEW_3 and VIEW_4 object also start with the ST_REV parameter. ST_REV is not listed in the commScript but it has to be considered when calculating the length of the view object and the number of elements of the view object.

In the device description for each block there is one VARIABLE_LIST covering all VIEW_3 elements and one VARIABLE_LIST covering all view 4 elements. Multiple VIEW_3 and VIEW_4 sections in the commScript are multiple sections in the VARIABLE_LIST. The sections in the VARIABLE_LIST are separated by predefined elements VL_ST_REV_01, VL_ST_REV_02, VL_ST_REV_03, …

Example:

The VARIABLE_LIST separator between the first and second VIEW_3 or VIEW_4 object is VL_ST_REV_01:

```
VL_ST_REV_01,              PARAM.ST_REV ;
```

If your VARIABLE_LIST consists of three VIEW_3 or VIEW_4 object a second separator is needed:

```
VL_ST_REV_02,              PARAM.ST_REV ;
```

4.6.4.3 Section FF_TRANSDUCER_BLOCK

Per default a commDevice transducer block has default 14 parameters (see Appendix 140). Device-specific parameters must be added. Only the device-specific parameters are listed in the commScript.

View lists with device-specific parameter follow the parameter list. The standard view parameters required by the FF specification are added automatically and don't have to be specified here. For details on the transducer block views see Appendix 141.

Parameter description

Item	Description
Default block tag	Block tag length: 4 to 16 octets. No leading underscores. Allowed characters: 'A'-'Z', 'a'-'z', '0'-'9', '_' Each transducer block must have a unique default block tag. The default block tag combined with the serial number of the commModule MBP appears in the block header of the transducer block.

Syntax:

```
FF_TRANSDUCER_BLOCK <Default block tag>
<List of additional parameters>
    VIEW_1
     <List of parameter names>
    END_VIEW_1
    VIEW_2
     <List of parameter names>
    END_VIEW_2
    VIEW_3
     <List of parameter names>
    END_VIEW_3
    VIEW_3
     <List of parameter names>
    END_VIEW_3
    VIEW_4
     <List of parameter names>
    END_VIEW_4
    VIEW_4
     <List of parameter names>
    END_VIEW_4
    VIEW_4
     <List of parameter names>
    END_VIEW_4
END_FF_TRANSDUCER_BLOCK
```

Example:

```
FF_TRANSDUCER_BLOCK AI_TB
   STANDARD_PARAM PRIMARY_VALUE              Record    DS_FLOAT_S
   PARAM         PV_UNIT                     Simple    UNSIGNED16
   PARAM         HART_CFG_CHANGE_CTR         Simple    UNSIGNED16
   PARAM         HART_TAG_DESC_DATE    S W   Record    HART_TAG_DESC_DATE_STRUCT
   PARAM        HART_PV_INFO                Record    HART_PV_INFO_STRUCT
   PARAM        HART_DEVICE_INFO            Record    HART_DEVICE_INFO_STRUCT
   PARAM         HART_ADDITIONAL_DEV_STATUS  Array(4)   BIT_STRING(1)

   VIEW_1
     PRIMARY_VALUE
   END_VIEW_1

   VIEW_2
     PV_UNIT
     HART_CFG_CHANGE_CTR
   END_VIEW_2

   VIEW_3
     PRIMARY_VALUE
     HART_ADDITIONAL_DEV_STATUS
   END_VIEW_3

   VIEW_4
     HART_TAG_DESC_DATE
     HART_PV_INFO
     HART_DEVICE_INFO
   END_VIEW_4
END_FF_TRANSDUCER_BLOCK
```

4.6.5 Physical block and PA transducer blocks

4.6.5.1 Device-specific parameters

Parameter description

Item	Description
Parameter class	PARAM: A device-specific parameter. NULL_PARAM: A gap in the parameter list.
Parameter name	Visible string: 4 to 32 octets, no leading underscores. Parameter name must be identical to the name used in the device description.
Parameter flags (for commModule connected with HART or Modbus device)	W: Parameter is writeable. W_MAN: Parameter is writable in modes MAN and OOS. W_OOS: Parameter is writeable in mode OOS only. U: Parameter is not mapped to HART read/write commands or to Modbus registers. X: Parameter is writeable, but only the current value can be written. Noother value is permitted. Has to be combined with 'U' flag. S: Static Parameter. Parameter is stored non-volatile, and the static revision counter (ST_REV) is incremented when the parameter value changes. N: Parameter is stored non-volatile, but the static revision counter will not be incremented when the parameter value changes. D: Parameter is dynamic. It is not stored non-volatile. C: Constant parameter (read-only).
Object class	Simple, Record, Array In case of arrays the number of elements is part of the object class name, e.g. Array(5) - for an array with five elements.
Parameter type	<Data type name>|<Data struct name> In case of octet and visible strings the string length is part of the data type name, e.g. VISIBLE_STRING(16).
Default values and constant values	See Default and constant values 79 for accepted default value format. Array and record elements are separated by white spaces. If no default values are specified, data type specific initial values are used. See table with initial values in Chapter Default values and constant values 79.

Syntax:

PARAM <Parameter name> <Parameter flags> <Object class> <Parameter type> [optional] <default/constant values>

Example:

```
PARAM PRIMARY_VALUE                      Record     DS_FLOAT_S
PARAM PV_UNIT                  S W_OOS   Simple     UNSIGNED16
PARAM HART_ADDITIONAL_DEV_STATUS         Array(13)  BIT_STRING(1)
```

Example of default/const values

```
PARAM UNMAP_PAR_OSTRING3     U S W    Simple    OCTET_STRING(3)   10,11,12
PARAM TB_UNMAP_VSTRING8_AR   U S W    Array(2)  VISIBLE_STRING(3) "un1" "un2"
PARAM TB_UNMAP_RECORD        U S W    Record    HART_STRUCT       "Tag" 1 2 3
```

4.6.5.2 View objects

Each commDevice PA block has two or more view objects:

- View 1 – contains standard parameters of physical block (see Appendix 145) and transducer blocks (see Appendix 146) . Device-specific parameters can be added to transducer block view 1 but not to physical block view 1.
- View 2 – contains standard parameters of physical block (see Appendix) 145. No device-specific parameters can be added.
- View 3 – not supported by PA commDevices
- View 4 – not supported by PA commDevices
- View 5 – Optional. Contains device-specific parameters
- …
- View <n> – Optional. Contains device-specific parameters

The maximum length of one view object is 122 bytes. A view object must not have more than 32 elements.

A maximum number of 8 vendor-specific view objects is allowed (from View 5 to View 12).

4.6.5.3 Section PA_PHYSICAL_BLOCK

Per default the commDevice physical block has 38 parameters (see Appendix 144 for details). Device-specific parameters can be added. Only the device-specific parameters are listed in the commScript. View lists with device-specific parameters follow the parameter list.

Parameter description

Item	Description
Block tag	Block tag length: 4 to 16 octets. No leading underscores. Allowed characters: 'A'-'Z', 'a'-'z', '0'-'9', '_' Block tag of physical block is optional. If no block tag is specified, the standard tag PHYSICAL is used. The block tag is used to identify the block in the commScript. It does not appear in any commDevice parameter.
PARENT_CLASS	PROFIBUS PA parent class of a block, see Appendix 159 for supported keywords. Optional parameter, default value is 'UNUSED'.

Syntax:

```
PA_PHYSICAL_BLOCK [optional] <Block tag> [optional] PARENT_CLASS = <Parent_Class Keyword>
 <List of device-specific parameters>
 VENDOR_VIEW
  <List of parameter names>
 END_VENDOR_VIEW
END_PA_PHYSICAL_BLOCK
```

Example:

```
PA_PHYSICAL_BLOCK HART_PHY PARENT_CLASS = TRANSMITTER
  PARAM HART_DEV_IDENT                Record HART_CMD_0_STRUCT
  PARAM HART_MESSAGE             S W Simple VISIBLE_STRING(32)
  PARAM ACME_DEV_CHARACTERISTICS      Record ACME_DEV_CHARACT_STRUCT
  VENDOR_VIEW
    ACME_DEV_CHARACTERISTICS
  END_VENDOR_VIEW
END_PA_PHYSICAL_BLOCK
```

4.6.5.4 Section PA_TRANSDUCER_BLOCK

A commDevice PA transducer block has 8 default parameters (see Appendix 146 for details). Device-specific parameters have to be added. Only the device-specific parameters are listed in the commScript.

View lists with device-specific parameters follow the parameter list.

Parameter description

Item	Description
Block tag	Block tag length: 4 to 16 octets. No leading underscores. Allowed characters: 'A'-'Z', 'a'-'z', '0'-'9', '_' Each transducer block must have an unique block tag. The block tag is used to identify the block in the commScript. It does not appear in any PA commDevice parameter.
PARENT_CLASS	PROFIBUS PA parent class of a block, see Appendix 159 for supported keywords. Optional parameter, default value is 'UNUSED'.
CLASS	PROFIBUS PA class of a transducer block, see Appendix 159 (6.14.2) for list of supported keywords. Optional parameter, default value is 'UNUSED'.

Syntax:

PA_TRANSDUCER_BLOCK <Block tag> [optional] PARENT_CLASS = <Parent_Class Keyword> [optional]
CLASS = <Class Keyword>

<List of device-specific parameters>

VIEW_1
<List of parameter names>
END_View_1

VENDOR_VIEW
<List of parameter names>
END_VENDOR_VIEW

VENDOR_VIEW
<List of parameter names>
END_VENDOR_VIEW

END_PA_TRANSDUCER_BLOCK

Example:

```
PA_TRANSDUCER_BLOCK AI_TB  PARENT_CLASS=PRESSURE  CLASS=DIFFERENTIAL
  PARAM PRIMARY_VALUE                  Record   DS_FLOAT_S
  PARAM PV_UNIT                        Simple   UNSIGNED16
  NULL_PARAM
  NULL_PARAM
  PARAM HART_CFG_CHANGE_CTR            Simple   UNSIGNED16
  PARAM HART_TAG_DESC_DATE         S W Record   HART_TAG_DESC_DATE_STRUCT
  PARAM HART_PV_INFO                   Record   HART_PV_INFO_STRUCT
  PARAM HART_DEVICE_INFO               Record   HART_DEVICE_INFO_STRUCT
  PARAM HART_ADDITIONAL_DEV_STATUS     Array(4) BIT_STRING(1)
  VIEW_1
    PRIMARY_VALUE
  END_VIEW_1
END_PA_TRANSDUCER_BLOCK
```

4.6.6 Channels and units

4.6.6.1 Section CHANNEL_LIST

Every transducer block parameter used for mapping a process value to an input/output function block is addressed via a channel number. The commScript contains a list of all possible channel numbers supported by the device. Each channel list entry describes for which IO function block type the channel is used and which transducer block is referenced. The parameter containing the process value is specified by the transducer block tag and the parameter name.

An output channel optionally has a readback value signalling the current position of the actuator. An AI or AO channel has to specify the transducer block parameter that contains the unit of the process value. This can be a unit type (USIGN16 value) or a record using data structure DS_SCALE.

Parameter description

Item	Description
Channel number	Range: 1 to 65535 Device-specific; each channel must have unique channel number.
FB type	Type of related function block: AI, DI, AO, DO.
TB name	Name of the transducer block where PV , readback and unit parameter reside.
PV parameter	Transducer block parameter containing the channel-related process value.
Readback parameter	Transducer block parameter containing the channel-related read-back value. For AO and DO channels only. Optional; the device may not support read-back function.
Unit parameter	Transducer block parameter containing the unit of the process value. For AI and AO channels only.

Syntax:

CHANNEL_LIST

CHANNEL <Channel number> <FB type> <Default transducer block tag> <PV parameter> [conditional, optional] <Readback parameter> [conditional] <Unit parameter>

END_CHANNEL_LIST

Example:

```
CHANNEL_LIST
  CHANNEL  1  AI  AI_TB  PRIMARY_VALUE                    PV_UNIT
  CHANNEL  2  AI  AI_TB  SECONDARY_VALUE                  SV_UNIT
  CHANNEL  3  AI  AI_TB  TERTIARY_VALUE                   TV_UNIT
  CHANNEL  4  AI  AI_TB  QUATERNARY_VALUE                 QV_UNIT
  CHANNEL 20  AO  AO_TB  FINAL_VALUE FINAL_POSITION_VALUE FINAL_VALUE_RANGE
```

4.6.6.2 Section UNIT_LIST

It is necessary to define all unit codes that are supported by the device. An entry in the unit list starts with the keyword UNIT. It specifies the unit, the working range and all channels supporting this unit.

Parameter description

Item	Description
Unit keyword	For the list of all supported units see Appendix 150.
EU_100%	Engineering unit value representing the upper end of the working range.
EU_0%	Engineering unit value representing the lower end of the working range.
Channel list	List of AI and AO channels supporting this unit.

Syntax:

```
UNIT_LIST
    UNIT  <Unit keyword> <EU_100%> <EU_0%> <Channel list>
END_UNIT_LIST
```

Example:

```
UNIT_LIST

//       unit name         | EU_100% |  EU_0%   | Channel List
// ------------------------------------------------------------------------
UNIT     PASCAL             1500000    50000     1
UNIT     BAR                15         0.5       1
UNIT     MILLIBAR           15000      500       1
UNIT     VENDOR_UNIT_01     11250.923  375.03    1       // mmHg
UNIT     KELVIN             423.15     223.15    2
UNIT     CELSIUS            150.0      -50.0     2
UNIT     PERCENT            100.0      0.0       20

END_UNIT_LIST
```

HART vendor units

If a HART device is connected with the commModule the unit list may contain HART vendor units too. For the HART vendor units see section HART_VENDOR_UNIT_MAP 99.

Modbus vendor units

If a Modbus device is connected with the commModule the unit list may contain Modbus vendor units too. For the Modbus vendor units see section MODB_VENDOR_UNIT_MAP 105.

4.6.6.3 Section ACTUATOR_CHECK_BACK

Item	Description
Check back number	CHECK_BACK_xx with 01 <= xx <= 16
Check back type	Currently only LOCAL_OVERRIDE is supported
Channel number	Channel number as defined in the section CHANNEL_LIST. Only channels which are linked to an AO or DO function block with readback can be used.

Syntax:

ACTUATOR_CHECK_BACK

CHECK_BACK <Check back number> <Check back type> < Channel number>

END_ACTUATOR_CHECK_BACK

Example:

```
ACTUATOR_CHECK_BACK
  //     | Check back number | Check back type | Channel number
         CHECK_BACK_01       LOCAL_OVERRIDE         20
         CHECK_BACK_02      LOCAL_OVERRIDE         30
END_ACTUATOR_CHECK_BACK
```

4.6.7 Diagnostic and error conditions

The principle purpose of the Field Diagnostics is to create a single group of parameters to aggregate all device status and diagnostics so that a Host system can integrate this information into its infrastructure. Consistent with NAMUR NE-107 the diagnostics are broken down into four categories: Failure, Off-Specification, Maintenance and Check Function. Each of these categories share 32 conditions that can be defined by the device manufacturer. Each condition may be mapped or not mapped for each category.

Softing has reserved 5 conditions (COND_00 .. COND_04) for internal or protocol related reasons. Currently, COND_01 and COND_02 are used, while COND_00 is reserved within the FF protocol.

- COND_01 indicates a problem in the internal communication with the device application, i.e. failing HART or Modbus communication. No valid process values are available, the device is treated to be defective.
- COND_02 is used by the commDevice to signal issues in NV data storage. The device continues to operate correctly, but may loose configuration data at next power cycle or restart. However, a power cycle or restart may also solve the problem.
- COND_03 and COND_04 are currently not used.

Recommended Action

When a field diagnostic condition is active the resource block parameter FD_RECOMMEN_ACT describes by enumerated action what should be done to alleviate the condition.

The relation between field diagnostic condition and enumeration code is fix:

Condition	Recommended action code
COND_00	3
COND_01	4
COND_02	5
COND_03	6
…	…
COND_30	33
COND_31	34

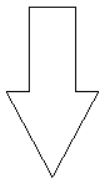
If multiple conditions are active at the same time the recommended action of the condition with the lowest number is shown.*

In the FF device description, you should adapt the recommended action texts according to the needs of your device. The recommended action texts are specified by the FF DD macro FD_RECOMMEN_ACT_ENUM.

* Please note that in FD-SW V3.40 in case of multiple conditions the condition with the highest number was shown.

4.6.7.1 Section FF_FDIAG_LIST

Parameter description

Item	Description
Condition number	FDIAG condition are mapped to FF field diagnostics parameter: COND_xx with 05 <= xx <= 31 COND_00 is predefined by FF standards (check condition) COND_01 is used by the commDevice to signal communication loss for the HART/Modbus serial communication COND_02 is used by the commDevice to signal issues in NV data storage. The device continues to operate correctly, but may loose configuration data at next power cycle or restart. COND_03 .. COND_04 are reserved for commDevice internal purposes
Condition group	According to NAMUR classification: MAINT_GOOD, MAINT_UNC, MAINT_BAD, CHECK_GOOD, CHECK_UNC, CHECK_BAD, OFFSPEC_UNC, OFFSPEC_BAD, FAIL_BAD
Sub-status	For UNCERTAIN and BAD quality the FF specifies sub-status. A subset of the sub-status is available: UNC_SUBST_VAL UNC_SUB_NORMAL UNC_EU_RANGE_VIOLATION UNC_SENSOR_CONF_NOT_ACC BAD_SENSOR_FAIL BAD_DEV_FAIL If multiple conditions are active at the same time the sub-status with the highest priority is set. lowest priority  highest priority
List of affected channels	The list of affected channels is a subset of the channels defined in the CHANNEL_LIST. In case of condition group MAINT_GOOD and CHECK_GOOD the list of affected channels is empty

Syntax:

FF_FDIAG_LIST

<Condition number> <Condition group> [optional]<Sub-status> [conditional|optional]<Affected channels>

…

END_FF_FDIAG_LIST

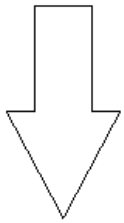
Example:

```
FF_FDIAG_LIST
  //     | Cond Group |   Sub-status                | Channel List
  // ------------------------------------------------------------
  COND_05  MAINT_GOOD
  COND_06  CHECK_GOOD
  COND_07  OFFSPEC_UNC                               1 2 3
  COND_08  OFFSPEC_UNC   UNC_SUBST_VAL               2 3 4
  COND_08  OFFSPEC_UNC   UNC_SENSOR_CONF_NOT_ACC     1 2 3 4
  COND_09  OFFSPEC_UNC   UNC_EU_RANGE_VIOLATION      2
  COND_10  OFFSPEC_UNC   UNC_SUB_NORMAL              1
END_FF_FDIAG_LIST
```

4.6.7.2 Section FF_EXT_DIAG_LIST

It is also possible to set the status and sub-status of a process value without setting a FF field diagnostic bit. This can be done with conditions 32 – 63. A separate section FF_EXT_DIAG_LIST can be used for setting the status and sub-status of process value.

Parameter description

Item	Description
Condition number	Extended diagnosis conditions are not mapped FF field diagnostics parameter: COND_yy; 32 <= yy <= 63
Sub-status	For extended diagnosis all sub-status of the FF field diagnostics are available. The extended diagnosis allows the following sub-status: UNC_NON_SPEC (lowest priority) UNC_INIT_VALUE UNC_SUBST_VAL UNC_SUB_NORMAL UNC_EU_RANGE_VIOLATION UNC_SENSOR_CONF_NOT_ACC BAD_NON_SPEC BAD_SENSOR_FAIL BAD_DEV_FAIL (highest priority)  If multiple conditions are active at the same time the sub-status with the highest priority is set.
List of affected channels	List of channels influenced by the diagnostics condition.

Syntax:

```
FF_EXT_DIAG_LIST
    <Condition number> <Sub-status> <Affected channels>
    …
END_FF_EXT_DIAG_LIST
```

Example:

```
// Diagnostic conditions, NOT mapped to FF Field Diagnostics parameters
// ---------------------------------------------------------------------
FF_EXT_DIAG_LIST
  //       | Sub-status       | Channel List
  // ------------------------------------
  COND_57   UNC_NON_SPEC       2 17
END_FF_EXT_DIAG_LIST
```

4.6.7.3 Section FF_RESB_ERROR_LIST

It is possible to map a diagnostic condition to a resource block error bit. This is done in section FF_RESB_ERROR_LIST. An entry is defined via its condition number (keyword COND_<i>, with i = 05 ..63) and the block error parameters defined by keyword RESB_<error bit>.

It is also possible to map a check back condition to a resource block error bit. In this case the error code RESB_ERR_LOC_OVERRIDE must be used.

Parameter description

Item	Description
Condition number	The following RESB error codes are available: RESB_ERR_MEM_FAIL, RESB_ERR_STATIC_DATA_LOST, RESB_ERR_NV_DATA_LOST, RESB_ERR_MAINT_SOON, RESB_ERR_MAINT_NOW
Check back number	The following RESB error code is available: RESB_ERR_LOC_OVERRIDE

Syntax:

FF_RESB_ERROR_LIST
<Condition number> <RESB error code>
<Check back number> < RESB_ERR_LOC_OVERRIDE>
END_FF_RESB_ERROR_LIST

Example:

```
FF_RESB_ERROR_LIST
  COND_53 RESB_ERR_NV_DATA_LOST
  CHECK_BACK_01 RESB_ERR_LOC_OVERRIDE
END_FF_RESB_ERROR_LIST
```

4.6.7.4 Section FF_TB_ERROR_LIST

It is possible to map a diagnostic condition to a transducer block error bit. This is done in section FF_TB_ERROR_LIST. An entry is defined via its condition number (keyword COND_<i>, with i = 05 .. 63), the name of the transducer block and the block error parameters defined by keyword TB_<error bit>.

Parameter description

Item	Description
TB error	The following transducer block error codes are available: TB_ERR_BLOCK_CFG, TB_ERR_LOC_OVERRIDE, TB_ERR_SENSOR_FAIL, TB_ERR_OUTPUT_FAIL, TB_ERR_READBACK_FAIL In case of TB_ERR_BLOCK_CFG the transducer block switches to OOS and the status of the PV and readback parameters switch to BAD|OOS. In case of TB_ERR_SENSOR_FAIL the status of the PV parameters switches to BAD_SENSOR_FAIL. In case of TB_ERR_OUTPUT_FAIL or TB_ERR_READBACK_FAIL the status of the readback parameters switch to BAD_DEVICE_FAIL.

Syntax:

FF_TB_ERROR_LIST
    <Condition number> <Default block tag> <TB error>
END_FF_TB_ERROR_LIST

Example:

```
FF_TB_ERROR_LIST
  COND_08 ANALOG_INPUT_TB TB_ERR_SENSOR_FAIL
  COND_57 ANALOG_INPUT_TB TB_ERR_BLOCK_CFG
END_FF_TB_ERROR_LIST
```

4.6.7.5 Section FF_TB_XD_ERROR_LIST

It is also possible to map a diagnostic condition to a transducer XD error parameter. This is done in section FF_TB_XD_ERROR_LIST. An entry is defined via its condition number (keyword COND_<i>, with i = 05 .. 63), the name of the transducer block and the XD error code.

Parameter description

Item	Description
Default block tag	Default transducer block tag.
Enumeration code	The XD_ERROR is of type ENUMERATED (1). The codes 1 to 25 are defined by FF; the codes 26 to 255 can be used for device-specific error messages

Note

If an XD_ERROR occurs then in the BLK_ERROR is “Other” block error bit is set. If the XD_ERROR code returns to 0 then the “Other” block error bit is reset.

Syntax:

FF_TB_XD_ERROR_LIST
<Condition number> <Default block tag> <Enumeration code>
END_FF_TB_XD_ERROR_LIST

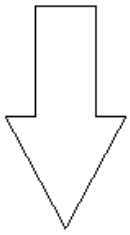
Example:

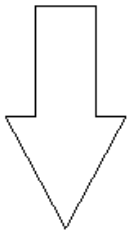
```
FF_TB_XD_ERROR_LIST
   COND_07 ANALOG_INPUT_TB 26
   COND_29 ANALOG_INPUT_TB 33
   COND_58 ANALOG_INPUT_TB 34
   COND_59 ANALOG_INPUT_TB 35
   COND_60 ANALOG_INPUT_TB 63
END_FF_TB_XD_ERROR_LIST
```

4.6.7.6 Section PA_DIAG_LIST

A commDevice supports 48 diagnostics conditions. Diagnostics conditions are mapped to the physical block parameter DIAGNOSIS_EXTENSION.

Parameter description

Item	Description
Condition ID	COND_xx with 05 <= xx <= 47 COND_01 is used by the commDevice to signal communication loss for the HART/Modbus serial communication. COND_02 is used by the commDevice to signal issues in NV data storage. The device continues to operate correctly, but may loose configuration data at next power cycle or restart. COND_03 .. COND_04 are reserved for commDevice internal purposes
Condensed status	According to PROFIBUS PA profile V3.02, table 79: GOOD GOOD_MAINT GODD_MAINT_DEMAND GOOD_FUNCT_CHECK UNC_PROCESS UNC_MAINT_DEMAND BAD_PROCESS BAD_MAINT_ALM BAD_FUNCT_CHECK lowest priority  highest priority If multiple conditions are active at the same time the condensed status with the highest priority is set. The condensed status implicitly defines a DIAGNOSIS bit that is set in condensed status mode (see table "Mapping Condensed Status to Condensed Diagnosis" below). The status values defined in the mapping are used as default status values. Due to the PROFIBUS PA specification it will be possible to set a different status value via the Diag_Event_Switch.Link_Status_Array parameter. See Chapter 6.1 (133) for details. The limit bits in the condensed status will be set if the keyword for classic status mapping is for low or high limit.
Classic diagnosis	According to PROFIBUS PA profile V3.02, table 69: DIA_HW_ELECTR, DIA_HW_MECH, DIA_TEMP_MOTOR, DIA_TEMP_ELECTR, DIA_MEM_CHKSUM, DIA_MEASUREMENT, DIA_NOT_INIT, DIA_INIT_ERR, DIA_ZERO_ERR, DIA_SUPPLY, DIA_CONF_INVAL, DIA_MAINTAINANCE, DIA_CHARACT This defines the bit of the DIAGNOSIS parameter to be set when the condition occurs. To indicate that a bit of DIAGNOSIS_EXTENSION is not mapped to DIAGNOSIS the additional value DIA_CLASSIC_NO_IND is supported.

Item	Description
Classic status	According to PROFIBUS PA profile V3.02, table 67: GOOD UNC_SUBST_VAL UNC_INIT_VALUE UNC_SENSOR_CONF_NOT_ACC UNC_EU_RANGE_VIOLATION UNC_SUB_NORMAL BAD_NON_SPEC BAD_CONFIG_ERR BAD_SENSOR_FAIL BAD_DEV_FAIL lowest priority  highest priority If multiple conditions are active at the same time the sub-status with the highest priority is set. For these conditions the limit bits in the status of the process value will be 0. Additional keywords: UNC_EU_RANGE_VIOLATION_HI_LIM UNC_EU_RANGE_VIOLATION_LO_LIM UNC_SENSOR_CONF_NOT_ACC_HI_LIM UNC_SENSOR_CONF_NOT_ACC_LO_LIM (can be used to set the low or high limit bits in the process value status)
List of affected PVs	The list of affected PVs is a subset of the PV parameters defined in the PA channel list.

Mapping *Condensed Status* to *Condensed Diagnosis*

Condensed Status	Condensed Diagnosis
GOOD	-
GOOD_MAINT	DIA_MAINTAINANCE
GOOD_MAINT_DEMAND	DIA_MAINTAINANCE_DEMAND
UNC_MAINT_DEMAND	DIA_MAINTAINANCE_DEMAND
BAD_MAINT_ALM	DIA_MAINTAINANCE_ALARM
UNC_PROCESS	DIA_INV_PRO_COND
BAD_PROCESS	DIA_INV_PRO_COND
GOOD_FUNCT_CHECK	DIA_FUNCTION_CHECK
BAD_FUNCT_CHECK	DIA_FUNCTION_CHECK

Note

This mapping from Condensed Status to Diagnosis is the default setting. It can be changed via the Diag_Event_Switch.Link_Status_Array. See Chapter 6.x (133) for details.

Syntax:

PA_DIAG_LIST
<Condition ID> <Condensed status> <Classic diagnosis> <Classic status> <List of affected PVs>
END_PA_DIAG_LIST

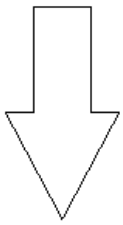
Example:

```
PA_DIAG_LIST
  COND_05   BAD_MAINT_ALM  DIA_HW_MECH         BAD_DEV_FAIL  1 2 3 4
  COND_06   GOOD_MAINT     DIA_CLASSIC_NO_IND  GOOD
END_PA_DIAG_LIST
```

4.6.7.7 Section PA_ADD_DIAG_LIST

The commScripter supports 16 additional diagnostic conditions. These are not mapped to DIAGNOSIS_EXT parameter and can be used for setting a DIAGNOSIS bit or the PV status value if classic status mode is selected.

Parameter description

Item	Description
Condition ID	COND_xx with 48 <= x <= 63
Classic diagnosis	According to profile V3.02, table 69: DIA_HW_ELECTR, DIA_HW_MECH, DIA_TEMP_MOTOR, DIA_TEMP_ELECTR, DIA_MEM_CHKSUM, DIA_MEASUREMENT, DIA_NOT_INIT, DIA_INIT_ERR, DIA_ZERO_ERR, DIA_SUPPLY, DIA_CONF_INVAL, DIA_MAINTAINANCE, DIA_CHARACT To indicate that a bit of DIAGNOSIS_EXTENSION is not mapped to DIAGNOSIS the additional value DIA_CLASSIC_NO_IND is supported.
Classic status	According to profile V3.02, table 57: GOOD (lowest priority) UNC_SUBST_VAL UNC_INIT_VALUE UNC_SUB_NORMAL UNC_EU_RANGE_VIOLATION UNC_SENSOR_CONF_NOT_ACC BAD_NON_SPEC BAD_CONFIG_ERR BAD_SENSOR_FAIL BAD_DEV_FAIL (highest priority)  If multiple conditions are active at the same time the sub-status with the highest priority is set.
List of affected PVs	The list of affected PVs is a subset of the PV parameters defined in the PA channel list. In case of status GOOD the list of related channels is empty.

Syntax:

PA_ADD_DIAG_LIST
<Condition ID> <Classic diagnosis> <Classic status> <List of affected PVs>
END_PA_ADD_DIAG_LIST

Example:

```
PA_ADD_DIAG_LIST
  COND_57  DIA_TEMP_ELECTR  UNC_SUB_NORMAL  1
END_PA_ADD_DIAG_LIST
```

4.6.8 Default values and constant values

This section is about default values of function blocks and constant values of resource, physical and transducer blocks.

The following table shows the supported data type and the supported format for default values and constant values:

Supported data types	Supported formats
UNSIGNED8	0 – 255, 0x00 – 0xFF
UNSIGNED16	0 – 65535, 0x00 – 0xFFFF
UNSIGNED32	0 – 4294967295, 0x00 – 0xFFFFFFFF
INTEGER8	-128 – 127
INTEGER16	-32768 – 32767
INTEGER32	-2147483648 – 2147483647
FLOAT	100.0 / -100.123
VISIBLE_STRING	This is a visible string. If the specified string is shorter than the specified string parameter then the remaining octets are filled with blanks (hex value 0x20)
OCTET_STRING	0x12,20,30,0x4A,0x4B,4C,11,22 An octet string is specified as a comma-separated list of values. Decimal and hexadecimal values are accepted
BIT_STRING(1)	One hex value: 0x10
BIT_STRING(2)	Two comma-separated hex values: 0x10,0x20
BIT_STRING(4)	Four comma-separated hex values: 0x10,0x20,0x20,0x10
DATE	year-month-day or year-month-dayThour:min:ms:SU year must be between 2000-2099 hour must be between 0-23 min must be between 0-59 ms must be between 0-59999 SU 0 (standard time) or 1 (Standard daylight saving time)

Supported data types	Initial values
UNSIGNED8	0
UNSIGNED16	0
UNSIGNED32	0
INTEGER8	0
INTEGER16	0
INTEGER32	0
FLOAT	0.0
VISIBLE_STRING(N)	N bytes filled with 0x20 (Whitespace)
OCTET_STRING(N)	N bytes filled with 0x00
BIT_STRING(1)	0x00
BIT_STRING(2)	0x00, 0x00
BIT_STRING(4)	0x00, 0x00, 0x00, 0x00
DATE	All fields set to 0

4.6.8.1 Section FF FBLK DEFAULT VALUE LIST

Optionally, you can specify default values for a small set of I/O function block parameters:

- CHANNEL AI, AO, DI, DO, blocks
- XD_SCALE AI, AO blocks

A function block is addressed via its block type and the number of the block. For example, if your device has four AI blocks, the third block is addressed as AI_3. The elements of the XD_SCALE parameter are specified using SUBIDX_<i>, i=1-4. If there are no device-specific default values the following initial values are used:

- CHANNEL 0
- XD_SCALE 100.0 0.0 0 0 (EU at 100%, EU at 0%, Units Index, Digits right of point)

Note

All FF hosts are able to work with the initial values. It is not necessary to specify device-specific default values.

Parameter description

Item	Description
Function block ID	The function block ID is composed of the function block type and an instances ID for each function block type. Instance ID: 1 to n AI_1, AI_2, … AI_n AO_1, AO_2, … AO_n DI_1, DI_2, … DI_n DO_1, DO_2, … DO_n
Parameter name	Name of the parameter. The following parameters are available: CHANNEL XD_SCALE
Sub-index	In case of XD_SCALE parameter a sub-index is required. SUBIDX_<i>, i = 1- 4
Value	Default value of the parameter

Syntax:

FF_FBLK_DEFAULT_VALUE_LIST

<Function block ID> <Parameter name> [conditional]<sub-index> <value>

END_FF_ FBLK_DEFAULT_VALUE_LIST

Example:

```
FF_FBLK_DEFAULT_VALUE_LIST
   AI_1 CHANNEL                1
   AI_1 XD_SCALE   SUBIDX_1    100.0    // EU at 100%
   AI_1 XD_SCALE   SUBIDX_2    0.0      // EU at 0%
   AI_1 XD_SCALE   SUBIDX_3    1342     // Units Index
   AI_1 XD_SCALE   SUBIDX_4    2        // digits right of the decimal point
   AI_2 CHANNEL                2
   AI_2 XD_SCALE   SUBIDX_1    100.0    // EU at 100%
   AI_2 XD_SCALE   SUBIDX_2    0.0      // EU at 0%
   AI_2 XD_SCALE   SUBIDX_3    1001     // Units Index
   AI_2 XD_SCALE   SUBIDX_4    1        // digits right of the decimal point
END_FF_FBLK_DEFAULT_VALUE_LIST
```

4.6.8.2 Section FF SCP FBLK DEFAULT VALUE LIST

You have to set the SCP mode configuration parameters for the SCP function blocks declared in section FF_FUNCTION_BLOCK_LIST.

Following function block parameters have to be configured in the commScript:

- CHANNEL AI, AO, DI, DO blocks
- UNIT AI, AO blocks

A SCP compatible function block is addressed via its block type and the number of the block. For example, if your device has four AI function blocks of which two are SCP compatible (as defined in section FF_FUNCTION_BLOCK_LIST), than the first two AI function blocks can be addressed as AI_1 and AI_2. It is mandatory to provide an entry for each configuration parameter of each SCP compatible function block.

Parameter description

Item	Description
Function block ID	The function block ID consists of the function block type and an instances ID for each function block type. Instance ID: 1 to n, where n is the number of SCP I/O block of one specific function block type AI_1, AI_2, … AI_n AO_1, AO_2, … AO_n DI_1, DI_2, … DI_n DO_1, DO_2, … DO_n
Parameter name	Name of the parameter. The following parameters are available: CHANNEL (for AI, AO, Di, DO) UNIT (for AI and AO)
Value	Value Default value of the parameter

Syntax:

```
FF_SCP_FBLK_DEFAULT_VALUE_LIST
    <Function block ID> <Parameter name> <Value>
END_FF_SCP_FBLK_DEFAULT_VALUE_LIST
```

Example:

```
FF_SCP_FBLK_DEFAULT_VALUE_LIST
// Block ID | CHANNEL/UNIT | Value
    AI_1 CHANNEL 1
    AI_1 UNIT CELSIUS
    AI_2 CHANNEL 2
    AI_2 UNIT PERCENT
    AO_1 CHANNEL 20
    AO_1 UNIT PERCENT
    DI_1 CHANNEL 17
    DO_1 CHANNEL 30
END_FF_SCP_FBLK_DEFAULT_VALUE_LIST
```

4.6.8.3 Section PA FBLK DEFAULT VALUE LIST

In difference to FF function blocks some PA function block parameters require device-specific default values:

- CHANNEL AI, AO, DI, DO blocks
- CHANNEL_UNIT AI block

A function block is addressed via its block type and the number of the block. For example, if your device has four AI blocks then the third block is addressed as AI_3. The elements of the OUT_SCALE parameter are specified using SUBIDX_<i>, i=1-4.

For each function block contained in the device the device-specific default value for parameter CHANNEL must be set with a valid channel number that you have defined in section CHANNEL_LIST.

CHANNEL_UNIT of AI blocks

The PV_SCALE is an array with two elements, EU_at_100% and EU_at_0%, describing the range of the process value. The unit of the process value does not appear in the AI function block but in the PV unit parameter of the associated transducer block.

The PV unit parameter of a channel is specified in the CHANNEL_LIST [66] section and the range of a unit (EU_at_100% and EU_at_0%) is specified in the UNIT_LIST [67] section.

With the CHANNEL_UNIT specified in the commScript the commDevice firmware sets the default values of the AI PV_SCALE and OUT_SCALE parameters. (According to PROFIBUS PA profile V3.02 the default values of PV_SCALE and OUT_SCALE have to be equal.)

If the PV unit parameter is mapped to a HART command or a Modbus register you have make sure that by default your HART/Modbus device provides the PV unit and PV range as specified in the commScript.

Optional default value for OUT_SCALE parameter of AO blocks

The PV_SCALE parameter of the AO function block is a record of type DS-36 [EU_at_100%, EU_at_0%, Unit Code, Valid digits after the point]. According to PROFIBUS PA profile V3.02 the default value of PV_SCALE is [100.0, 0.0, %, 2].

If the OUT_SCALE is not specified the commDevice firmware initializes the OUT_SCALE to [100.0, 0.0, %, 2]. If OUT_SCALE is specified in the commScript at least SUBIDX_1 to SUBIDX_3 have to be defined. Otherwise, the commScripter generates an error message.

Parameter description

Item	Description
Function block ID	The function block ID is composed of the function block type and an instances ID for each function block type. Instance ID: 1 to n AI_1, AI_2, … AI_n AO_1, AO_2, … AO_n …
Parameter name	Name of the parameter. The following parameters are available: CHANNEL CHANNEL_UNIT OUT_SCALE

Item	Description
Sub-index	In case of OUT_SCALE parameter a sub-index is required. SUBIDX_<i>, i = 1- 4
Value	Default value of the parameter

Syntax:

PA_FBLK_DEFAULT_VALUE_LIST

<Function block ID> <Parameter name> [conditional]<sub-index> <value>

END_PA_ FBLK_DEFAULT_VALUE_LIST

Example:

```
PA_FBLK_DEFAULT_VALUE_LIST
  AI_1 CHANNEL                1
  AI_1 CHANNEL_UNIT           BAR

  AI_2 CHANNEL                2
  AI_2 CHANNEL_UNIT           CELSIUS

  DI_1 CHANNEL                10

  AO_1 CHANNEL                20
  AO_1 OUT_SCALE   SUBIDX_1   40.0       // Upper end of range
  AO_1 OUT_SCALE   SUBIDX_2   0.0        // Lower end of range
  AO_1 OUT_SCALE   SUBIDX_3   CENTIMETER // Unit
  AO_1 OUT_SCALE   SUBIDX_4   2          // Valid digits right of the decimal point

  DO_1 CHANNEL                30

END_PA_FBLK_DEFAULT_VALUE_LIST
```

4.6.9 commModule with HART device

4.6.9.1 Section HART_COMM

This section specifies the HART communication settings. The baud rate is determined automatically by using CMD0 to scan for a HART slave response. Scan is started with the baud rate specified in the commScript.

Parameter description

Item	Description
Baudrate	Integer value: 0, 1200, 2400, 4800, 9600, 19200, 38400, 57600 Default value: 0 = Start baud rate scan with 1200 Baud
Parity	String value: “Odd”, “Even”, “NoParity” Default value: “Odd”
StopBits	Number of Stop Bits. Integer Value: 1, 2 Default value: 1
Handshake	Using RTS Signal or not: String value: “On”, “Off” Default value: “On”
RTS	Polarity of RTS when active: String value: “Low”, “High” Default value: “Low”
RTSSetupTime	Time of setting RTS to active before HART frame transmission is started. Unit: Bit times, range 0 – 200. Default value: 0
InterframeGap	Minimum time between two HART frames: Unit: Character times, range: 2 – 60. Default value: 0 -> Firmware selects HART default value
SlaveTimeout	Wait time for response from HART slave. Unit: Character times, range: 10 – 200. Default value: 0 -> Firmware uses HART default timeout value STO
CycleTimeMin	Minimum time between HART commands to read the process values given in ms. Range is 0 – 5000. Default value is 0 ms.

Syntax:

<Item> = <Integer Value>
<OR>
<Item> = “<String value>”
HART_COMM
<List of HART settings>
END_ HART_COMM

Example:

```
HART_COMM
  Baudrate  = 1200
  Parity    = "Odd"
  StopBits  = 1
  DataBits  = 8
  Handshake = "On"
  RTS       = "Low"
END_HART_COMM
```

Note

It is not necessary to specify a keyword if the default value can be used.

4.6.9.2 Section HART_CMD_MAPPING

This Section specifies the HART commands and their mapping to FF/PA block parameters. Command numbers 0-30 and 32-255 are supported. Command numbers do not have to be unique and can be used for multiple command descriptions. Typically, multiple command 9 descriptions can be used to read different sets of device variables.

A HART command is either of type READ which reads out values from the HART device and updates the referred transducer block parameter or it is of type WRITE. Write commands are used when writeable transducer or resource/physical block parameters are written or when an output value is calculated by a function block and updated in the transducer block.

For every command three sections are defined:

- CMD_REQ describes the format/data used for sending HART request
- CMD_RES describes the format/data used for a HART response from the device
- CMD_RES_CODES list of possible response codes and mapping to error/warning

Each CMD_REQ/CMD_RES contains the definition of the HART data type and the FF/PA block parameter. A record must specify the sub-index of the record element. Not all data items of a HART response must be mapped to FF/PA block parameters. If a data item is not mapped the CMD_RES entry contains only the HART data type.

An entry for the CMD_RES_CODES contains keyword RES_CODE followed by the numerical value of the response code followed by the error/warning category which could be one of the following:

- HART_SUCCESS
- HART_PARAM_CHECK
- HART_WRITE_PROTECT
- HART_WARNING
- HART_TEMP_ERROR

Data type mapping

The following table shows how HART data types are mapped to FF/PA data types:

HART	FF	PA	Note
BIT_STRING(1)	BIT_STRING(1)	OCTET_STRING(1)	DD: BIT_ENUMERATED (1)
BIT_STRING(2)	BIT_STRING(2)	OCTET_STRING(2)	DD: BIT_ENUMERATED (2)
BIT_STRING(4)	BIT_STRING(4)	OCTET_STRING(4)	DD: BIT_ENUMERATED (4)
DATE			Must be mapped to three consecutive UNSIGNED record or array elements. In the command description the sub-index of the first element (day) has to be specified.
TIME	UNSIGNED32	UNSIGNED32	
ENUM(1)	UNSIGNED32	UNSIGNED32	DD: ENUMERATED (1)
ENUM(2)	UNSIGNED32	UNSIGNED32	DD: ENUMERATED (2)
FLOAT	FLOAT	FLOAT	
LATIN(2)	VISIBLE_STRING(2)	VISIBLE_STRING(2)	
LATIN(119)	VISIBLE_STRING(119)	VISIBLE_STRING(119)	Supported length: 2 … 119 (read/write parameters)

HART	FF	PA	Note
LATIN(122)	VISIBLE_STRING(122)	VISIBLE_STRING(122)	Supported length: 2 … 122 (read only parameters)
LATIN(120)	VISIBLE_STRING(120)	VISIBLE_STRING(120)	
PACKED(3)	VISIBLE_STRING(4)	VISIBLE_STRING(4)	
PACKED(6)	VISIBLE_STRING(8)	VISIBLE_STRING(8)	
PACKED(87).(rw)	VISIBLE_STRING(116).(rw)	VISIBLE_STRING(116).(rw)	Supported length: 4, 8 … 116 octets (read/write parameters) [len % 4 == 0]
PACKED(90).(ro)	VISIBLE_STRING(120).(ro)	VISIBLE_STRING(120).(ro)	Supported length: 4, 8 … 120 octets (read-only parameters) [len % 4 == 0]
UNSIGNED5	UNSIGNED8	UNSIGNED8	The most significant five bits are mapped to the least significant five bits of UNSIGN8
UNSIGNED8	UNSIGNED8	UNSIGNED8	
UNSIGNED16	UNSIGNED16	UNSIGNED16	
UNSIGNED24	UNSIGNED32	UNSIGNED32	The most significant byte is skipped
UNSIGNED32	UNSIGNED32	UNSIGNED32	
INTEGER8	INTEGER8	INTEGER8	
INTEGER16	INTEGER16	INTEGER16	
INTEGER24	INTEGER32	INTEGER32	
INTEGER32	INTEGER32	INTEGER32	
UNIT	UNSIGNED16	UNSIGNED16	DD: ENUMERATED (2) Unit codes are mapped from FF/PA to HART and vice versa. See also Chapter commScript unit codes 150.
STATUS	UNSIGNED8	UNSIGNED8	This data type is not used for direct mapping. It can be used to indicate that the field contains status information. It is always necessary that this data type is mapped to a SUBIDX_STATUS of a process value from the channel list for input blocks or the readback value for output blocks. Please see HART device variable status further down in this section for detailed information on using the keyword STATUS.

Data item

Parameter description

Item	Description
Data type	See table above.
Block tag	Block tag as defined in the resource, physical or transducer block section.
Sub-index	Sub-index specifies one element in a record or an array SUBIDX_1 to SUBIDX_n. For the data structures DS_FLOAT_S and DS_DISC_S the special SUBIDX_PV has to be used for addressing the process value.
CONST	A HART request frame may contain constant values. The constant value must be encoded in 2 octets (1 or 2 octets (UNSIGNED8 or UNSIGNED16)

Syntax:

<Data type> <Block tag> <Parameter name> [conditional]<Sub-index>
<OR>
<Data type> CONST <constant value>

Example:

```
UNIT     AITB_1    PRIMARY_VALUE_UNIT
FLOAT    AITB_1    PRIMARY_VALUE        SUBIDX_PV
UNSIGNED8 CONST     35
```

Read commands

All response codes different from SUCCESS are expected to be temporary read errors. If the HART device is permanently unable to provide the requested data then the HART device is expected to raise a maintenance alarm.

The following table shows which data items can be used in the request. If NONE is given, no other request item shall be specified. Other items – CONST or unmapped parameters – can be combined.

Parameter description for CMD_REQ

Item	Description
NONE	NONE specifies an empty list of data items
CONST item	<Datatype> CONST <Value
UNMAPPED parameter item	<Datatype> <BlockTag> <ParameterName> [<Subindex>]. Note 1: Using a parameter in the Read Request is permitted only for parameter of type SYN (synchronous access). Note 2: The definition of the parameter in the transducer block or resource/physical block should have the U flag. If this is not the case the commScripter treats the parameter as UNMAPPED, so no mapping must be defined for this parameter.

Syntax:

READ_CMD <Command number>
CMD_REQ
<List of data item>
CMD_RES
<List of data item>
CMD_RES_CODES
<List of response codes>
END_READ_CM

Example 1:

```
// ------------------------------------------------
// Read command 0 (Read Unique Identifier) ------
// ------------------------------------------------
READ_CMD 0
 CMD_REQ // Request
  NONE

 CMD_RES // Response data
  UNSIGNED8 // '254' not mapped
  ENUM(2) HART_RES HART_DEV_IDENT SUBIDX_1 // 'Device type'
  UNSIGNED8 // 'Min preambles' not mapped
  UNSIGNED8 // 'HART rev' not mapped
  UNSIGNED8 HART_RES HART_DEV_IDENT SUBIDX_2 // 'Dev rev'
  UNSIGNED8 HART_RES HART_DEV_IDENT SUBIDX_3 // 'SW rev'
```

```
  UNSIGNED5 HART_RES HART_DEV_IDENT SUBIDX_4 // 'HW rev' - 'Phy. Sign. Code' // can
not
                                                 be mapped
  BIT_STRING(1) // 'Flags' is not mapped
  UNSIGNED24 HART_RES HART_DEV_IDENT SUBIDX_5 // 'Device ID'
  UNSIGNED8 // 'Min preambles' not mapped
  UNSIGNED8 // 'Max device vars' not mapped
  UNSIGNED16 AI_TB HART_CFG_CHANGE_CTR // 'Cfg change ctr'
  BIT_STRING(1) // 'Ext status' not mapped
  ENUM(2) // 'Manufac ident' not mapped
  ENUM(2) // 'Private label' not mapped
  ENUM(1) // 'Device profile' not mapped

 CMD_RES_CODES // Response codes
END_READ_CMD
// End of read command 0 ------------------------
```

Line **`UNSIGNED8 HART_RES HART_DEV_IDENT SUBIDX_2 // 'Dev rev`**' means that 'Device Revision Level' is mapped to device-specific resource block (block tag: HART_RES) parameter HART_DEV_IDENT, sub-index 2

Example 2:

```
// ----------------------------------------------
// Read command 178 that is parameterized  ------
// ----------------------------------------------
READ_CMD 178
 CMD_REQ // Request
  UNSIGNED16 AI_TB  DATA_SET_SELECTOR
  UNSIGNED8  CONST  2

 CMD_RES // Response data
  UNSIGNED16 // selector is returned
  UNSIGNED32 AI_TB DATA_SET SUBIDX_1
  UNSIGNED32 AI_TB DATA_SET SUBIDX_2
  UNSIGNED32 AI_TB DATA_SET SUBIDX_3
  UNSIGNED32 AI_TB DATA_SET SUBIDX_4

 CMD_RES_CODES // Response codes
END_READ_CMD
// End of read command 178 ------------------------

Using the following parameter definition:
PARAM   DATA_SET_SELECTOR        U   W     Simple  UNSIGNED16       0
PARAM   DATA_SET                 SYN       Array(4)  UNSIGNED32
```

HART device variable status

Some HART response data contain the values and status of device variables (command 9 response data). It is possible to use the HART status of device variables to affect the status and limit of an FF/PA process value. Furthermore, the device variable status can be used to trigger conditions as shown in section HART_DIAG_MAPPING (95). The HART common tables Revision 25.0 specification defines the status code for the device variables in the command 9 response as follows.

Bit Number	Description
7-6	Process Data Status. Overall status of the device variable value. 3 Good 2 Manual/Fixed 1 Poor Accuracy 0 Bad
5-4	Limit Status. Indicates whether the device variable is responding to process changes. 3 Constant 2 High Limited 1 Low Limited 0 Not Limited
3	More Device Variable Status Available. Set if expanded Device Family Status Command contains diagnostic information that is useful to the Host Application. Must be reset to zero is a Device Variable does not support any Device Family.
2-0	Device Family Specific Status. Specified by corresponding a Device Family. Must be reset to zero is Device Variable does not support any Device Family.

Mapping of device variable status to process variable status

As the meaning of bit 7-6 is predefined in the HART specification it is per default used to set the FF/PA status of a process value. However, it is possible to disable this implicit mapping by specifying the IGN_STATUS keyword in HART_DIAG_MAPPING section

The following implicit mapping is used from HART to FF status:

HART Device Variable Status Mapping

HART Status	FF Status
0xC0	Good_NonCascade (0x80)
0x80	Readback value for Output Blocks (AO/DO): Good_Cascade_LocalOverride (0xD8) Process value for Input Blocks (AI/DI): Uncertain_LastUsableValue (0x48)
0x40	Uncertain_SensorConversionNotAccurate (0x50)
0x00	Bad_SensorFailure (0x10)

Limit Bits

Bit 4-5 determine the limit bits in the HART device variable status and can be mapped to the FF/PA limit bits in the status of a process variable. The meaning of the limit bits is identical in HART and FF/PA but in FF/PA bit 0-1 are used as limit bits.

Limit information in HART Device Variable Status

Limit value	Description
0	No limit
1	Low limit
2	High limit
3	Constant

Explicit Mapping

It is possible to define explicit condition mapping for bit 0-3 of a HART device variable status or for the values BAD, UNCERTAIN, MANUAL in the same way as bits of CMD 48 response data can be used. This is explained in detail in section HART_DIAG_MAPPING.

Definition of the HART Device Status Mapping
The mapping of device variable status to a process variable parameter status can be achieved by using the Keyword STATUS in the command response of a read request:

Syntax:

STATUS <Transducer block name> <process variable parameter> SUBIDX_STATUS

Example:

```
READ_CMD 9
  CMD_REQ // Request
    UNSIGNED8   CONST       246  // Slot 0: Device Variable Code for
PV
    UNSIGNED8   CONST       247  // Slot 1: Device Variable Code for
SV
  CMD_RES // Response data
    UNSIGNED8                    // Extended Field Device Status
    UNSIGNED8                    // Slot 0: Device Variable Code
    UNSIGNED8                    // Slot 0: Device Variable
Classification
    UNIT        AI_TB       PV_UNIT
    FLOAT       AI_TB       PRIMARY_VALUE         SUBIDX_PV
    STATUS      AI_TB       PRIMARY_VALUE         SUBIDX_STATUS
    UNSIGNED8                   // Slot 1: Device Variable Code
    UNSIGNED8                   // Device Variable Classification
    UNIT        AI_TB       SV_UNIT
    FLOAT       AI_TB       SECONDARY_VALUE       SUBIDX_PV
    UNSIGNED8
  CMD_RES_CODES  // Response codes
    RES_CODE 8  HART_WARNING
 END_READ_CMD
CHANNEL_LIST
  CHANNEL 1   AI AI_TB  PRIMARY_VALUE    PV_UNIT
  CHANNEL 2   AI AI_TB  SECONDARY_VALUE  SV_UNIT
END_CHANNEL_LIST
```

This example shows that the process value, process value status and the related unit for channel 1 are retrieved from the HART command 9 response data. The HART device variable status is used to set the status for parameter PRIMARY_VALUE. For channel 2 - SECONDARY_VALUE - only the process value and the related unit is retrieved from command 9 response data. The HART device variable status is not used in this example.

Write commands

Write response codes:

- HART_SUCCESS
  Write command was executed successfully. Response code '0 – Success' needs not to be specified in the commScript. HART_SUCCESS has to be used if a response code different from '0' shall trigger a positive FF/PA write response.
- HART_WARNING
  HART warnings are not evaluated by the commModule MBP firmware. A HART warning results in a positive FF write response
- HART_PARAM_CHECK
  The written value was not valid. A negative FF/PA write response is sent.
- HART_WRITE_PROTECT
  HART write protect mode is enabled. A negative FF/PA write response is sent.
- HART_TEMP_ERROR
  Value(s) are temporarily not writeable. A negative FF/PA write response is sent.

All response codes different from HART_SUCCESS, HART_WARNING, HART_PARAM_CHECK and HART_WRITE_PROTECT are expected to be temporary write errors. If the HART device is permanently unable to store the new data then the HART device is expected to raise a maintenance alarm.

HART devices may send write response data. In case of HART_SUCCESS and HART_WARNING the response data are stored in the RAM of the commModule MBP.

Syntax:

WRITE_CMD <Command number>
CMD_REQ
<List of data item>
CMD_RES
<List of data item>
CMD_RES_CODES
<List of response codes>
END_WRITE_CMD

Example:

```
WRITE_CMD 18
  CMD_REQ // Request
    PACKED(6)    AI_TB    HART_TAG_DESC_DATE  SUBIDX_1
    PACKED(12)   AI_TB    HART_TAG_DESC_DATE  SUBIDX_2
    DATE         AI_TB    HART_TAG_DESC_DATE  SUBIDX_3
  CMD_RES // Response data
    PACKED(6)    AI_TB    HART_TAG_DESC_DATE  SUBIDX_1
    PACKED(12)   AI_TB    HART_TAG_DESC_DATE  SUBIDX_2
    DATE         AI_TB    HART_TAG_DESC_DATE  SUBIDX_3
  CMD_RES_CODES  // Response codes
    RES_CODE  6   HART_PARAM_CHECK
    RES_CODE  7   HART_WRITE_PROTECT
    RES_CODE  9   HART_PARAM_CHECK
    RES_CODE 16   HART_PARAM_CHECK
END_WRITE_CMD
```

Additionally to user defined parameters, it is possible to provide a mapping for the resource block parameters SOFTWARE_REV and HARDWARE_REV and for the physical block parameters SOFTWARE_REVISION and HARDWARE_REVISION. This constitutes an alternative to defining these parameters in the DEVICE_SETTINGS section of the commScripts. The two revisions can be referenced in any read command as demonstrated:

Example:

```
READ_CMD 177
 CMD_REQ // Request
  NONE
 CMD_RES // Response data
  PACKED(24)  HART_RES   SOFTWARE_REV
  LATIN(24)  HART_RES   HARDWARE_REV
 CMD_RES_CODES  // Response codes
  RES_CODE 8  HART_WARNING
END_READ_CMD
```

The length of the parameter is determined by the length of the HART data. The supported data types are LATIN and PACKED.

4.6.9.3 Section FACTORY_RESET_CMD

The factory reset command is issued when the FF *Restart-with-factory-defaults* or the PA FACTORY_RESET command is performed.

- *Restart-with-factory-defaults* is optional for FF devices. If your device does not support a factory-reset command it is strongly recommended to remove the RESTART_FACTORY_DEFAULT code from the RESTART macro in the DD file: *Tools\DD\cs_common\ff_standard_param.h*
- For PROFIBUS PA devices the FACTORY_RESET command is mandatory.
- No response data.
- Response code is expected to be '0 – Success'. Each result different from '0 – Success' is expected to activate a maintenance alarm.

Syntax:

FACTORY_RESET_CMD <Command number>
CMD_REQ
<List of data item>
END_FACTORY_RESET_CMD

Example:

```
// -------------------------------------------
// Factory reset command 143 ------------------
// -------------------------------------------
FACTORY_RESET_CMD 143
CMD_REQ // Request
NONE
// No response data
// No response code different from 0
END_FACTORY_RESET_CMD
// Factory reset command 143 ------------------
```

4.6.9.4 Section HART_DIAG_MAPPING

Command #48 bits can be mapped to commScript diagnostic conditions. 64 diagnostic conditions are supported. Conditions 00 to 04 have a predefined meaning and cannot be mapped to application related events. For FF devices see Sections FF_DIAG_LIST [70] and FF_EXT_DIAG_LIST [71]) and for PA devices see Sections PA_DIAG_LIST [75] and PA_ADD_DIAG_LIST [78]).

Parameter description

Item	Description
HART_CMD	The current version supports command 48 only.
Byte number	Byte number in the HART response data.
Bit number	Number of the bit indicating the diagnostics condition.

Syntax:

HART_DIAG_MAPPING
<Condition ID> <HART_CMD> <Byte number> <Bit number>
END_HART_DIAG_MAPPING

Example:

```
HART_DIAG_MAPPING
    COND_05 CMD_48 BYTE_0 BIT_0
    COND_06 CMD_48 BYTE_0 BIT_1
    COND_07 CMD_48 BYTE_0 BIT_2
    COND_08 CMD_48 BYTE_0 BIT_3
    COND_09 CMD_48 BYTE_0 BIT_4
    COND_10 CMD_48 BYTE_0 BIT_5
    COND_11 CMD_48 BYTE_0 BIT_6

    COND_12 CMD_48 BYTE_1 BIT_0

    COND_36 CMD_48 BYTE_4 BIT_4
    COND_38 CMD_48 BYTE_4 BIT_7

    COND_57 CMD_48 BYTE_7 BIT_1
END_HART_DIAG_MAPPING
```

Mapping of HART device variable status to diagnostic conditions

In addition to the implicit setting of the process value status certain bits or bit groups of the device variable status can be mapped to diagnostic condition COND_05 - COND_63. To do this, the status of the process value has to be mapped by using keyword STATUS as shown in section HART_CMD_MAPPING [86].

It is possible to map the device family status (bits 0-3) and the status quality (bits 7-6) from a device variable status to a diagnostic condition by using following syntax:

Syntax

HART_DIAG_MAPPING
<Condition ID> CHANNEL_STATUS _<Channel number> <Mapping Keyword >
END_HART_DIAG_MAPPING

Where the channel number represents the id of the channel associated to the process variable as defined in the CHANNEL_LIST section and the mapping keyword is one of the keywords as shown in table below.

Mapping Keyword	Description
BIT_0	This attribute indicates that the condition will be active if the device family status bit 0 is set.
BIT_1	This attribute indicates that the condition will be active if the device family status bit 1 is set.
BIT_2	This attribute indicates that the condition will be active if the device family status bit 2 is set.
BIT_3	This attribute indicates that the condition will be active if the device family status bit 3 is set.
USE_BAD	This attribute indicates that the condition will be active if the status quality indicates BAD. The mapped condition hat to be of type BAD (DeviceVariableStatus & 0xC0 == 0x00)
USE_UNC	This attribute indicates that the condition will be active if the status quality indicates POOR ACCURACY. The mapped condition hat to be of type UNCERTAIN (DeviceVariableStatus & 0xC0 == 0x40)
USE_MAN	This attribute indicates that the condition will be active if the device variable status indicates MANUAL (DeviceVariableStatus & 0xC0 == 0x80)

Disabling implicit mapping of Device Variable status mapping
By default, the status and the limit bits of a mapped device variable are mapped to the status of the process variable. Furthermore, in case of process variables which represent readback parameters of Analog Output or Discrete Output function blocks, a status quality of Manual/Fixed will set the block into local override mode

This behavior can be disabled if the device variable status is the status value shall be used only to trigger diagnosis conditions:

Syntax

HART_DIAG_MAPPING
 COND_EXT CHANNEL_STATUS _<Channel number> <Ignore Keyword >
END_HART_DIAG_MAPPING

Ignore Keyword	Description
IGN_STATUS	Per default the status information in bits 6-7 will be used to set the status of the related FF/PA process variable and possibly set the associated output function block in local override mode. By using this attribute it is possible to disable this i.e. to ignore the information in bit 6-7
IGN_LIMIT	Per default the limit bits (4-5) in the device variable status will be used to set the limit bits in the FF/PA process value. By using this attribute the limit bits from the HART device variable status will be ignored.

Example:

```
HART_DIAG_MAPPING

  COND_05 CMD_48 BYTE_0 BIT_5
  COND_06 CMD_48 BYTE_0 BIT_6
  ...

  COND_10 CHANNEL_STATUS_01 BIT_0
  COND_11 CHANNEL_STATUS_01 BIT_1
  COND_12 CHANNEL_STATUS_01 BIT_2
  COND_13 CHANNEL_STATUS_01 BIT_3
  COND_14 CHANNEL_STATUS_01 USE_BAD
  COND_15 CHANNEL_STATUS_01 USE_UNC
  COND_16 CHANNEL_STATUS_01 USE_MAN

  COND_EXT CHANNEL_STATUS_01 IGN_STATUS
  COND_EXT CHANNEL_STATUS_01 IGN_LIMIT

END_HART_DIAG_MAPPING
```

4.6.9.5 Section HART_CHECK_BACK_MAPPING

Command #48 bits can be mapped to commScript check back conditions. 16 check back conditions are supported. See section ACTUATOR_CHECK_BACK 68. It is possible to use a certain bit both in CHECK_BACK as well as HART_DIAG_MAPPING.

Parameter description

Item	Description
COND_<cond_nr>	Condition number that is mapped
CMD_<cmd_nr>	Number of HART command used to read the status information. Currently only CMD_48 is supported
Byte number	Byte number in the HART response data.
Bit number	Number of the bit indicating the diagnostics condition.

Syntax:

```
HART_CHECK_BACK_MAPPING
  <Condition ID> <HART_CMD> <Byte number> <Bit number>
END_HART_CHECK_BACK_MAPPING
```

Example:

```
HART_CHECK_BACK_MAPPING
  CHECK_BACK_01 CMD_48 BYTE_6 BIT_1
  CHECK_BACK_02 CMD_48 BYTE_6 BIT_2
END_HART_CHECK_BACK_MAPPING
```

Note

It is possible to define multiple entries for a certain condition. This means the same condition can be triggered by different reasons.

4.6.9.6 Section HART_VENDOR_UNIT_MAP

In addition to the standard unit codes device-specific unit codes can be defined.

Parameter description

Item	Description
Vendor unit keyword	20 vendor-specific unit codes are available: VENDOR_UNIT_170 to VENDOR_UNIT_179 and VENDOR_UNIT_240 to VENDOR_UNIT_249
FF unit code	All valid FF unit codes.
PA unit code	All valid PA unit codes. If no PA unit code is specified the commScripter assumes that the FF unit code and the PA unit code are identical.

Syntax:

HART_VENDOR_UNIT_MAP
<Vendor unit keyword> <FF unit code> [conditional]<PA unit code>
END_HART_VENDOR_UNIT_MAP

Example:

```
HART_VENDOR_UNIT_MAP

//               | FF unit code | PA unit code
VENDOR_UNIT_170   1282           1283
VENDOR_UNIT_171   32768
VENDOR_UNIT_179   1285
VENDOR_UNIT_240   1286
VENDOR_UNIT_249   1287

END_HART_VENDOR_UNIT_MAP
```

4.6.10 commModule with Modbus device

4.6.10.1 General

Modbus addresses

Some conventions govern how access to Modbus entities (input registers, holding registers) is referenced. It is important to make a distinction between entity number and entity address:

- Entity numbers combine entity type and entity location within their description table
- Entity address is the starting address, a 16-bit value in the data part of the Modbus frame. As such its range goes from 0 to 65,535

In the traditional standard, numbers for those entities start with a digit, followed by a number of four digits in range 1–9,999. This translates into addresses between 0 and 9,998 in data frames.

The register types in commScripts are determined by the keywords 'INP' for input register and 'HLD' for holding register. Therefore it is possible to use the complete address range 0 - 65535 as a register address.

Byte and word ordering

The Modbus specification doesn't define exactly how the data is stored in the registers. Therefore, some manufacturers implemented Modbus in their equipment to store and transmit the higher byte first followed by the lower byte. Alternatively, others store and transmit the lower byte first.

Similarly, when registers are combined to represent 32-bit data types, some devices store the higher 16 bits (high word) in the first register and the remaining low word in the second while others do the opposite.

It doesn't matter which order the bytes or words are sent in, as long as the receiving device knows which way to expect it.

For example, the float value 42.43 (Hex: 42 29 B8 52 (big endian) is stored in two 16-bit registers:

- 4229 B852 high byte first high word first "big endian"
- B852 4229 high byte first low word first
- 2942 52B8 low byte first high word first
- 52B8 2942 low byte first low word first "little endian"

4.6.10.2 Modbus functions supported by commDevice firmware

The commDevice firmware implements a Modbus RTU master. The Modbus master supports the following Modbus functions.:

- Read Input Registers 4
- Read Holding Registers 3
- Write Single Holding Register 6
- Write Multiple Holding Registers 16

4.6.10.3 Section MODB_COMM

The commDevice implements a Modbus RTU master. It is necessary to configure the serial communication parameters and the byte/word ordering.

Parameter description

Item	Description
SlaveAddress	1 to 247
Baudrate	Integer value: 1200, 2400, 4800, 9600, 19200, 38400, 57600, 115200
Parity	String value: “Odd”, “Even”, “NoParity” Optional, default value: “NoParity”
StopBits	Number of Stop Bits. Integer Value: 1, 2 Optional, default value: 1
ByteWordOrder	Byte and word order for 32-bit figures: “HB-HW” High-Byte first High-Word first / big-endian “HB-LW” High-byte first Low-Word first “LB-HW” Low-byte first High-Word first “LB-LW” Low-byte first Low-Word first / little-endian There is no separate byte order for 16-bit figures. If Low-Byte first is selected for 32-bit figures then Low-Byte first is used for 16-bit figures too. Otherwise High-Byte first is used for 16-bit figures. The selected byte and word order also applies to the bit-string data types. The selected byte and word order does not apply to octet and visible strings. The bytes are stored as sequence: <Byte-1, Byte-2> <Byte-3, Byte-4> <Byte-5, … Optional, default value: “HB-HW”
InterframeGap	Minimum time between two Modbus frames. Integer value: 10..1000. Unit: bit-time Optional, default value is 20 bit-times
ResponseTimeout	Maximum response delay of the Modbus slave. Integer value: 100-10000. Unit: bit-time.
CycleTimeMin	Minimum time between Modbus read requests to read the process values, given in ms. Range is 0 – 5000. Optional, default value is 0 ms.

Syntax:

<Item> = <Integer Value>

<OR>

<Item> = “<String value>”

Example:

```
MODB_COMM
   SlaveAddress    = 1
   BaudRate        = 38400
   Parity          = "Odd"
   StopBits        = 1
   ByteWordOrder   = "HW-HB"
   ResponseTimeout = 40
END_MODB_COMM
```

4.6.10.4 Section MODB_PARAM_MAP

ModbusRegister specifies the address of the first Modbus register of n consecutive registers. Every Modbus register keeps two characters and _xx specifies the length of the parameter for FF and PA. If the value for _xx is greater than 16, than only 16 registers will be used for a pa device. If no length is specified, the maximum length is used i.e. 32 for FF and 16 for PA.

It is recommended to place the parameters in continuous register if possible in order to minimize the number of Modbus read accesses that are necessary. Process values and status condition are read with a higher priority. So these can be in a separate register range.

For parameters with attribute SYN it is required that a continuous register range is used because the Modbus read request will be sent when the fieldbus read request is performed and the fieldbus response is delayed until the Modbus response is received. Using multiple Modbus requests could cause timeouts for the fieldbus request.

Data types

The following data types are supported:

FF/PA data type	Number of registers	Note
UNSIGNED8	1	FF DD: UNSIGNED_INTEGER (1), ENUMERATED (1)
UNSIGNED16	1	FF DD: UNSIGNED_INTEGER (2), ENUMERATED (2)
UNSIGNED32	2	FF DD: UNSIGNED_INTEGER (4)
INTEGER8	1	
INTEGER16	1	
INTEGER32	2	
FLOAT	2	
OCTET_STRING	1 to 60	
VISIBLE_STRING	1 to 60	Recommendation: Number of registers = 1 - 16
BIT_STRING(1)	1	FF DD: BIT_ENUMERATED (1)
BIT_STRING(2)	1	FF DD: BIT_ENUMERATED (2)
BIT_STRING(4)	2	FF DD: BIT_ENUMERATED (4)

Parameter description

Item	Description
Register type	INP, HLD
Register address	Range: 0 - 65535 See note above: Modbus addresses 100
Block tag	Block tag as defined in the resource, physical or transducer block section.
Sub-index	Sub-index specifies one element in a record or an array SUBIDX_1 to SUBIDX_n. For data structures DS_FLOAT_S and DS_DISC_S the special SUBIDX_PV has to be used for addressing the process value.

Syntax:

MODB_PARAM_MAP
<Register type> <Register address> <Block tag> <Parameter name> [conditional]<Sub-index>
END_MODB_PARAM_MAP

Example:

```
MODB_PARAM_MAP // ---------------------------------------------------------------
 INP       0   MODB_TB     PRIMARY_VALUE            SUBIDX_PV
 INP       2   MODB_TB     SECONDARY_VALUE          SUBIDX_PV
 INP       4   MODB_TB     PRIMARY_VALUE_D          SUBIDX_PV
 INP       5   MODB_TB     FINAL_POSITION_VALUE     SUBIDX_PV
 INP       7   MODB_TB     FINAL_POSITION_VALUE_D   SUBIDX_PV
 INP       8   MODB_RES    DEV_IDENT
 HLD       0   MODB_TB     FINAL_VALUE              SUBIDX_PV
 HLD       2   MODB_TB     FINAL_VALUE_D            SUBIDX_PV
 HLD       3   MODB_TB     PV_UNIT
 HLD       4   MODB_TB     SV_UNIT
END_MODB_PARAM_MAP
```

Additionally to user defined parameters, it is also possible to provide a mapping for the resource block parameters SOFTWARE_REV and HARDWARE_REV and for the physical block parameters SOFTWARE_REVISION and HARDWARE_REVISION. This constitutes an alternative to defining these parameters in the DEVICE_SETTINGS section of the commScripts.

Syntax:

<ModbusRegisterType> <ModbusRegisterAddress> <Resource/Physical Block Tag> SOFTWARE_REV[_xx]
<ModbusRegisterType> <ModbusRegisterAddress> <Resource/Physical Block Tag> HARDWARE_REV[_xx]

Example:

```
MODB_PARAM_MAP // ---------------------------------------------------------------
…
INP             200       MODB_RES     SOFTWARE_REV
INP             216       MODB_RES     HARDWARE_REV
…
END_MODB_PARAM_MAP
```

4.6.10.5 Section MODB_FACTORY_RESET

- The factory reset register is written when the FF Restart-with-factory-defaults or the PA FACTORY_RESET command is performed.
- Restart-with-factory-defaults is optional for FF devices. If your device does not support a factory-reset-command it is strongly recommended to remove the RESTART_FACTORY_DEFAULT code from the RESTART macro in the DD file: *Tools\DD\cs_common\ff_standard_param.h*
- For PROFIBUS PA devices the FACTORY_RESET command is mandatory.
- The commModule firmware does not evaluate the response data issued by the Modbus device.

Parameter description

Item	Description
Register type	HLD
Register address	Range: 0 - 65535 See note above: Modbus addresses 100
Factory reset code	0x0000 to 0xFFFF

Syntax:

MODB_FACTORY_RESET
HLD <Register address> <Factory reset code>
END_ MODB_FACTORY_RESET

Example:

```
MODB_FACTORY_RESET
 HLD      5                42
END_MODB_FACTORY_RESET
```

4.6.10.6 Section MODB_STD_UNIT_MAP

The commScripter supports a list of standard unit codes 150. This list is derived from the HART engineering unit code tables. As Modbus does not specify any unit codes all unit codes used by the Modbus device must be mapped to FF or PA unit codes.

Parameter description

Item	Description
Unit keyword	Valid range: unit keywords are listed in Section Unit Codes 150.
Modbus unit code	Range: 0 to 0xFFFF

Syntax:

MODB_STD_UNIT_MAP
<FF/PA unit keyword> <MODB unit code>
END_MODB_STD_UNIT_MAP

Example:

```
MODB_STD_UNIT_MAP
KELVIN          1000
CENTIMETER      1012
PERCENT          555
inH20_68F       8000
END_MODB_STD_UNIT_MAP
```

4.6.10.7 Section MODB_VENDOR_UNIT_MAP

The commScripter supports up to 10 vendor-specific unit codes which are not covered by the list of standard unit codes.

Parameter description

Item	Description
Vendor unit keyword	10 vendor-specific unit codes are available: VENDOR_UNIT_01 to VENDOR_UNIT_10
MODB unit code	Range: 0 to 0xFFFF
FF or PA	All valid FF or PA unit codes

Syntax:

MODB_VENDOR_UNIT_MAP
<Vendor unit keyword> <MODB unit code> <FF or PA>
END_MODB_VENDOR_UNIT_MAP

Example:

```
 MODB_VENDOR_UNIT_MAP
//              | MODB unit code | FF unit code | PA unit code
  VENDOR_UNIT_01  10250           1608           1558        // mg/L
END_MODB_VENDOR_UNIT_MAP
```

4.6.10.8 Section MODB_DIAG_MAP

The commScripter supports 59 device-specific diagnostics conditions. Conditions 00 to 04 have a predefined meaning and cannot be mapped to application related events. See Section FF_DIAG_LIST [70] and FF_EXT_DIAG_LIST [71] and for PROFIBUS PA devices PA_DIAG_LIST [75] and PA_ADD_DIAG_LIST [78].

Parameter description

Item	Description
Register type	INP
Register address	Range: 0 - 65535 See note above: *Modbus addresses* [100]
Bit number	BIT_n / Range: 0 to 15 BIT_0: 0x0001 / BIT_15: 0x8000

Syntax:

MODB_DIAG_MAP
<Condition ID> INP <Register address> <Bit number>
END_MODB_DIAG_MAP

Example:

```
MODB_DIAG_MAP
COND_05  INP  100  BIT_7
COND_23  INP  100  BIT_9
COND_31  INP  101  BIT_5
END_MODB_DIAG_MAP
```

4.6.10.9 Section MODB_CHECK_BACK_MAP

The commScripter supports 16 device-specific check back condition events. See section ACTUATOR_CHECK_BACK [68]. It is possible to define the same bit in MODB_CHECK_BACK_MAP and in MODB_DIAG_MAP.

Parameter description

Item	Description
Register type	INP
Register address	Range: 0 - 65535 See note above: *Modbus addresses* [100]
Bit number	BIT_n / Range: 0 to 15 BIT_0: 0x0001 / BIT_15: 0x8000

Syntax:

MODB_CHECK_BACK_MAP
<Condition ID> INP <Register address> <Bit number>
END_MODB_CHECK_BACK_MAP

Example:

```
MODB_CHECK_BACK_MAP
   CHECK_BACK_01 INP        44         BIT_5
   CHECK_BACK_02 INP        40         BIT_6
END_MODB_CHECK_BACK_MAP
```

4.7 Flashing a commScript configuration table

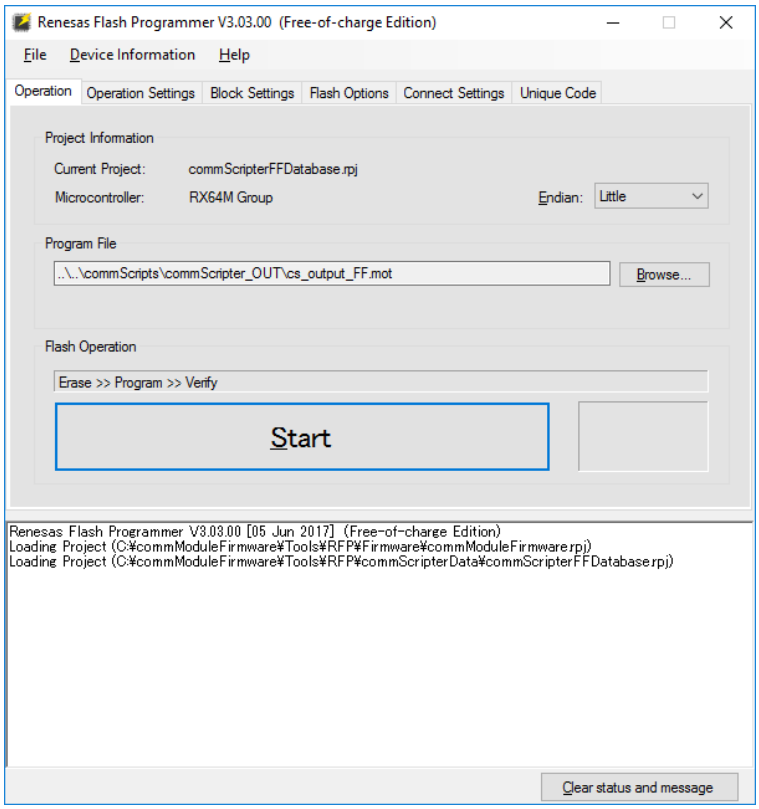
The Renesas Flash Programmer (RFP) is used to load the commScript configuration database to the commModule MBP flash. Click here to download the latest version of the tool from the Renesas website.

Note
The following instructions are an example of how to flash an FF configuration table. To flash a PA configuration table select the PA file names and directories given in brackets.

Note
The following project files are intended to be used with a Renesas E1 emulator. The same RFP projects are available if a Renesas E2 Lite emulator will be used instead of the E1. These projects are in the subfolder RFP_E2Lite.

Note
With version 3.60 it is also possible to write / update the commScript configuration table via fieldbus using Firmware Update mechanisms. See chapter Firmware download file 124.

1. Select **File → Open Project** and browse to file *RFP\commScripterData\commScripterFFDatabase.rpj* (use *RFP\commScripterData\commScripterPADatabase* for PA configuration table) in the delivery. All subsequent starts of the RFP will open this project file automatically.



The SREC file *cs_output_FF.mot* is given in the Program File line.

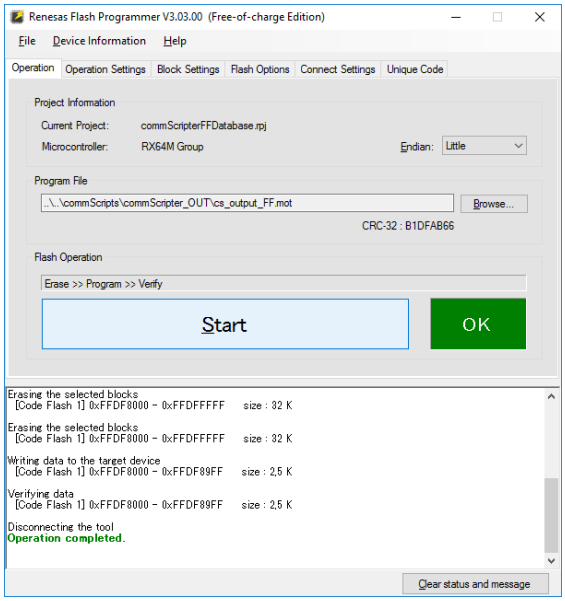
2. Click **[Browse]** to select the *cs_output_FF.mot*. (Select *cs_output_PA.mot* for PA configuration.)
The RFP project is set up to erase only the required flash blocks. So you will only need to connect the Renesas emulator to the PC running RFP via USB and to the commModule MBP using the JTAG

connector of the emulator. As the commModule MBP is powered by the emulator it must not be connected to the H1 fieldbus.

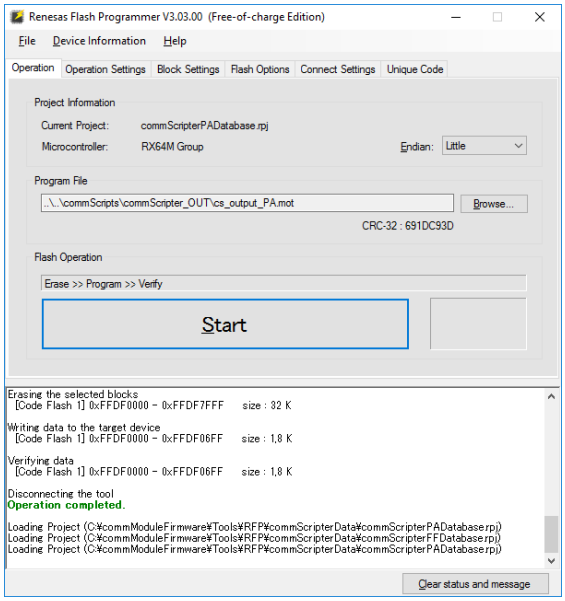
3. Click **[Start]** when a connection is established to flash the content of *cs_output_FF.mot* to the commModule MBP. A progress window is shown until the flash action is finished. When the *cs_output_FF.mot* file has been successfully flashed, the **[OK]** button is shown green. The status window below shows the log entries for erasing, writing and verifying the selected flash blocks.

4. Disconnect the emulator from the commModule MBP and connect it to the H1 fieldbus.

FF example



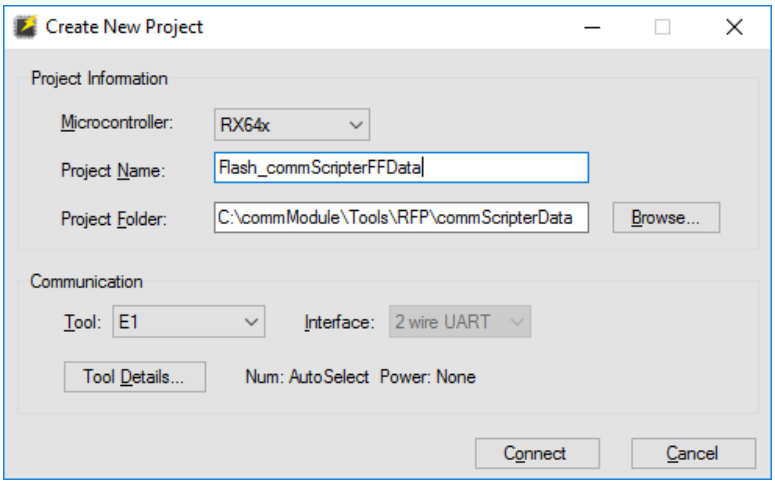
PA example



Usually Renesas provides backward compatibility for RFP project files. So it should be possible to use the project file also in a new RFP version. If this is not the case the following steps show how to manually setup the RFP to program the *cs_output_FF.mot*. or *cs_output_PA.mot* .

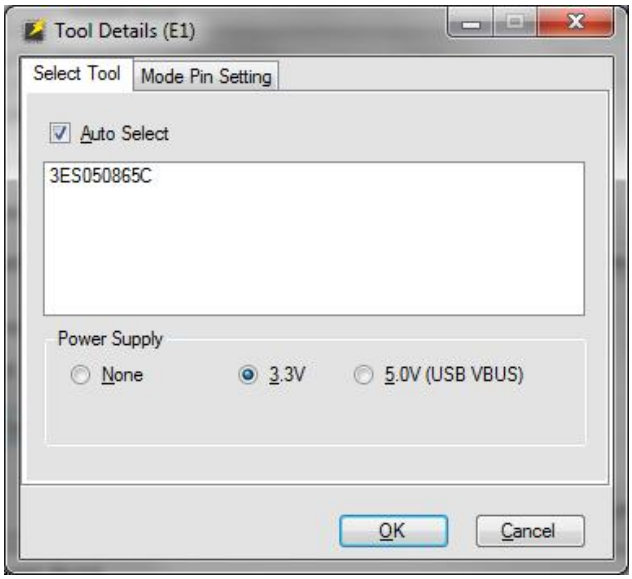
1. Select **File → Create New Project**.

2. Select microcontroller RX64M, select a local directory as project folder and enter a project name as shown below:



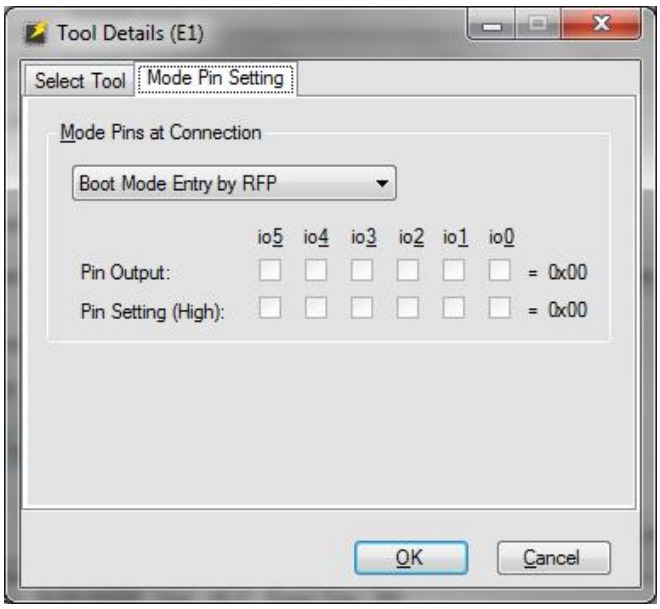
3. Select Renesas E1 as tool to be used for the connection to the commModule MBP and click **[Tool Details]** for further options.

4. Select Power Supply 3.3V by using the checkbox. you will need to connect one emulator over USB so that Auto Select will be selected and the serial number of the connected emulator is displayed. If multiple emulators are connected, select the required emulator.



5. Select the Mode Pin Setting and select the preferred settings.

6. Press **[OK]** and then Connect.



Note

The E1 configuration menu is also accessible after the RFP project has been created via the Connect Setting tab.

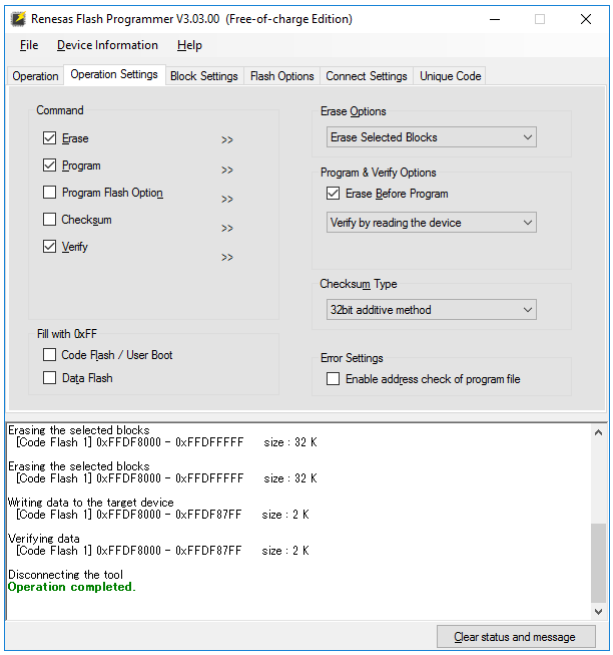
7. Open the Operation Setting tab.

8. Tick the checkboxes for Erase, Program and Verify.

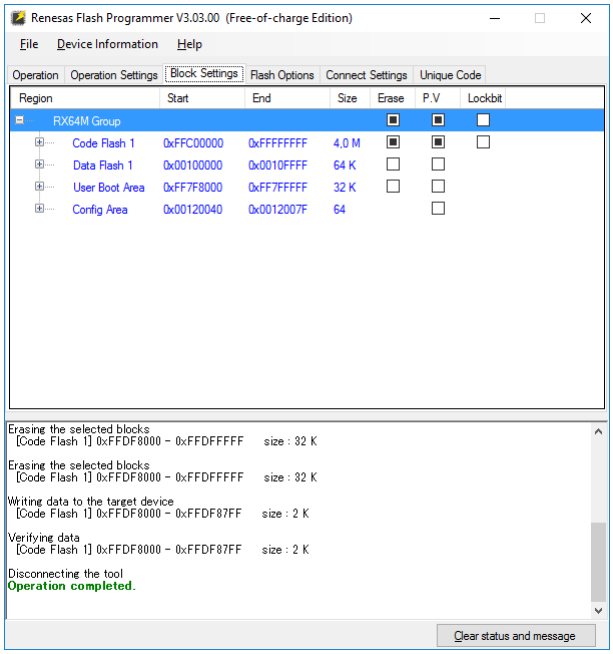
9. Select **Erase Option → Erase Selected Blocks.**

WARNING

If **Erase All Blocks** or **Erase Chip** option is selected as Erase Option the firmware and production data will be erased and the commModule MBP cannot be operated any longer!

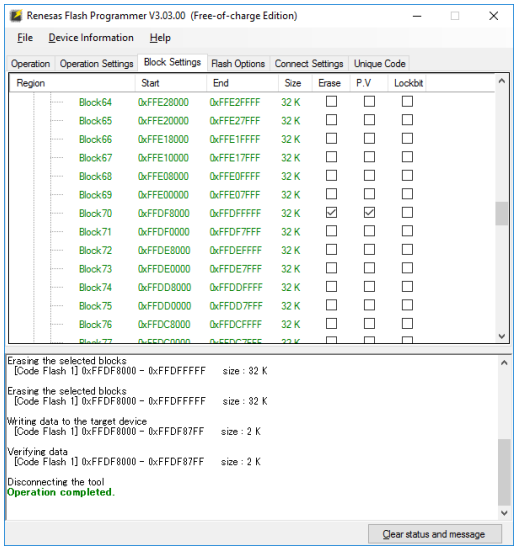


10. Change to the Block Setting tab and deselect all checkboxes for Erase, P.V. and Lockbit.

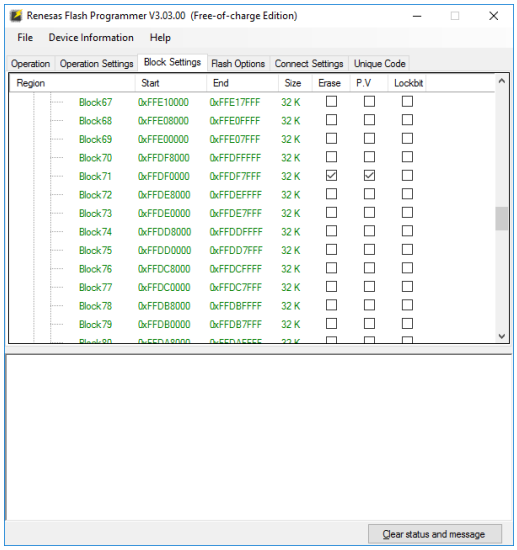


11. Expand the Code Flash 1 and select Erase and P.V. action for block 70 starting at 0xFFDF8000 for the commScripter FF configuration data table and block 71 starting at 0xFFDF0000 for the commScripter PA configuration data table.

PA example



FF example



WARNING
Selecting other blocks may delete the firmware!

4.8 Importing DD binaries

The following files are needed to import a device description into an FF host system:

- <DeviceRevision><DDRevision>.sy5 generated by FF Tokenizer
- <DeviceRevision><DDRevision>.ff5 generated by FF Tokenizer
- <DeviceRevision><DDRevision>01.cff generated by commScripter

Older FF host systems may require these files:

- <DeviceRevision><DDRevision>.sym generated by FF Tokenizer
- <DeviceRevision><DDRevision>.ffo generated by FF Tokenizer
- <DeviceRevision><DDRevision>01.cff generated by commScripter

Note

Before importing a device description make sure that the DD and CFF files are located in the same folder.

5 Deliverables

5.1 commScripter

The *commScripter.exe* and the commScripter utility *DLL CS_Utility.dll* can be found in directory *Tools**commScripter*.

5.2 commScripter Start HART (STH) Demo Device

The following sections describe the deliverables for commScripter STH device (FF device type 0320). It contains the commScripter input files and a batch to invoke the commScripter. In addition the DD source files are contained. The FF DD source files are grouped in a list of common files that can be used for all devices and a list of sources files specific for the STH device. The output files generated by the FF Tokenizer are also provided as well as the CFF file that is created by commScripter.

5.2.1 FF DD files

Common DD input files:

The following files are DD files which have been run by the FF Tokenizer and do not need to be modified.

- Tools\DD\cs_common\cs_ai.ddl
- Tools\DD\cs_common\cs_ao.ddl
- Tools\DD\cs_common\cs_ar.ddl
- Tools\DD\cs_common\cs_block_errdsc.ddl
- Tools\DD\cs_common\cs_cs.ddl
- Tools\DD\cs_common\cs_di.ddl
- Tools\DD\cs_common\cs_diag_tb.ddl
- Tools\DD\cs_common\cs_diag_tb.h
- Tools\DD\cs_common\cs_do.ddl
- Tools\DD\cs_common\cs_is.ddl
- Tools\DD\cs_common\cs_pid.ddl
- Tools\DD\cs_common\cs_resb.ddl
- Tools\DD\cs_common\cs_resb.h
- Tools\DD\cs_common\cs_sc.ddl
- Tools\DD\cs_common\ff_standard_param.ddl
- Tools\DD\cs_common\ff_standard_param.h

Device-specific DD input files:

The following files are specific for the Start HART device.

- Tools\DD\cs_start_hart_0320\cs_start_hart.ddl
- Tools\DD\cs_start_hart_0320\cs_start_hart.h
- Tools\DD\cs_start_hart_0320\cs_start_hart_fblks.ddl
- Tools\DD\cs_start_hart_0320\cs_start_hart_resb.ddl
- Tools\DD\cs_start_hart_0320\cs_start_hart_resb.h
- Tools\DD\cs_start_hart_0320\cs_hart_tb.ddl
- Tools\DD\cs_start_hart_0320\cs_start_hart_tb.h
- Tools\DD\cs_start_hart_0320\cs_demo_tb.ddl
- Tools\DD\cs_start_hart_0320\cs_demo_tb.h
- Tools\DD\cs_start_hart_0320\tok_cs_start_hart.bat

5.2.2 FF DD/CFF files

The following files are generated by the FF Tokenizer based on the input files described in the section *DD Files* above:

- Tools\DD\0320\0601.ff5 DD binary file (V5 format)
- Tools\DD\0320\0601.sy5 DD symbol file (V5 format)
- Tools\DD\0320\0601.ffo DD binary file (V4 legacy format)
- Tools\DD\0320\0601.sym DD symbol file (V4 legacy format)
- Tools\DD\0320\symbols.txt Ids of DD symbols

The CFF (FF common file format) file generated by the commScripter provides an offline description of the device blocks and parameters:

- Tools\DD\0320\060101.cff

For demonstration purposes a FDI package is included for the Demo device:

- Tools\DD\0320\Softing_AG.CS_START_HART.1.0.0.FF.fdix

5.2.3 PA EDD/GSD files

The following EDD files are specific for the Start HART device:

- Tools\EDD\0320\Softing_Sample_0320.devices
- Tools\EDD\0320\0117_0320_01.ddl
- Tools\EDD\0320\0117_0320_01_macros.h

- Tools\EDD\0320\0117_0320_01_menus.inc
- Tools\EDD\0320\softing_macros.h
- Tools\EDD\0320\phy_std.inc
- Tools\EDD\0320\phy_ext.inc
- Tools\EDD\0320\funcX_AI.inc
- Tools\EDD\0320\transX_std.inc
- Tools\EDD\0320\transX_diag.inc
- Tools\EDD\0320\trans1_ext.inc
- Tools\EDD\0320\trans1_demo.inc
- Tools\EDD\0320\pa_status.h
- Tools\EDD\0320\pa_units.h
- Tools\EDD\0320\prefix.h
- Tools\EDD\0320\rsp_codes.h
- Tools\EDD\0320\standard.dct

The GSD file, generated by commScripter, contains the characteristic communication features of the STH device.

- Tools\GSD\0320\SOFT1169.gsd

5.2.4 commScripts

The following script files need to be processed in the commScripter tool with the DD binary files generated by the FF Tokenizer:

- Tools\commScripts\STH_0320\cs-start-hart.cct (main script file)
- Tools\commScripts\STH_0320\cs-start-hart-mapping.cct
- Tools\commScripts\STH_0320\cs-start-hart-transducer.cct
- Tools\commScripts\STH_0320\cs-start-hart-values.cct
- Tools\commScripts\STH_0320\cs-softing-demo-transducer.cct

The following batch files can be used to invoke the commScripter and create flashable SREC file and CFF file for the STH device:

- Tools\commScripts\Cr8_STH_output.bat
- Tools\commScripts\Cr8_PA_STH_output.bat

5.3 commScripter HART Device

The following sections describe the deliverables for commScripter HART device (FF device type 0307). It contains the commScripter input files and a batch to invoke the commScripter. In addition the DD source files are contained. The FF DD source files are grouped in a list of common files that can be used for all devices and a list of DD sources files specific for the HART device. The output files generated by the FF Tokenizer are also provided as well as the CFF file that is created by commScripter.

5.3.1 FF DD files

For the common DD input files click here (113).

Device-specific DD input files:

The following files are specific for the HART device.

- Tools\DD\cs_hart_0307\cs_hart.ddl
- Tools\DD\cs_hart_0307\cs_hart.h
- Tools\DD\cs_hart_0307\cs_hart_ai_tb.ddl
- Tools\DD\cs_hart_0307\cs_hart_ai_tb.h
- Tools\DD\cs_hart_0307\cs_hart_ao_tb.ddl
- Tools\DD\cs_hart_0307\cs_hart_ao_tb.h
- Tools\DD\cs_hart_0307\cs_hart_di_tb.ddl
- Tools\DD\cs_hart_0307\cs_hart_di_tb.h
- Tools\DD\cs_hart_0307\cs_hart_do_tb.ddl
- Tools\DD\cs_hart_0307\cs_hart_do_tb.h
- Tools\DD\cs_hart_0307\cs_hart_fblks.ddl
- Tools\DD\cs_hart_0307\cs_hart_resb.ddl
- Tools\DD\cs_hart_0307\cs_hart_resb.h
- Tools\DD\cs_hart_0307\tok_cs_hart.bat

5.3.2 FF DD/CFF files

The following files are generated by the FF Tokenizer based on the input files described in the section *DD Files* above:

- Tools\DD\0307\0601.ff5 DD binary file (V5 format)
- Tools\DD\0307\0601.sy5 DD symbol file (V5 format)
- Tools\DD\0307\0601.ffo DD binary file (V4 legacy format)
- Tools\DD\0307\0601.sym DD symbol file (V4 legacy format)
- Tools\DD\0307\symbols.txt Ids of DD symbols

The CFF (FF common file format) file generated by the commScripter provides an offline description of the device blocks and parameters.

- Tools\DD\0307\060101.cff

5.3.3 PA EDD/GSD files

The following EDD files are specific for the HART device:

- Tools\EDD\0307\Softing_Sample_0307.devices
- Tools\EDD\0307\0117_0307_01.ddl
- Tools\EDD\0307\0117_0307_01_macros.h
- Tools\EDD\0307\0117_0307_01_menus.inc
- Tools\EDD\0307\softing_macros.h
- Tools\EDD\0307\phy_std.inc
- Tools\EDD\0307\phy_ext_msg.inc
- Tools\EDD\0307\funcX_AI.inc
- Tools\EDD\0307\funcX_AO.inc
- Tools\EDD\0307\funcX_DI.inc
- Tools\EDD\0307\funcX_DO.inc
- Tools\EDD\0307\transX_std.inc
- Tools\EDD\0307\transX_diag.inc
- Tools\EDD\0307\trans1_ext_cplx.inc
- Tools\EDD\0307\trans2_ext.inc
- Tools\EDD\0307\trans3_ext.inc
- Tools\EDD\0307\trans4_ext.inc
- Tools\EDD\0307\pa_status.h
- Tools\EDD\0307\pa_units.h
- Tools\EDD\0307\prefix.h

- Tools\EDD\0307\rsp_codes.h
- Tools\EDD\0307\standard.dct

The GSD file, generated by commScripter, contains the characteristic communication features of the HART device.

- Tools\GSD\0307\SOFT3807.gsd

5.3.4 commScripts

The following script files need to be processed in the commScripter tool with the DD binary files generated by the FF Tokenizer:

- Tools\commScripts\HART_0307\cs-hart.cct (main script file)
- Tools\commScripts\HART_0307\cs-hart-mapping.cct
- Tools\commScripts\HART_0307\cs-hart-transducers.cct
- Tools\commScripts\HART_0307\cs-hart-values.cct

The following batch files can be used to invoke the commScripter and create flashable SREC file and CFF file for the HART device:

- Tools\commScripts\Cr8_HART_output.bat
- Tools\commScripts\Cr8_PA_HART_output.bat

5.4 commScripter Modbus Device

The following sections describe the deliverables for commScripter Modbus device (FF device type 0309). It contains the commScripter input files and a batch to invoke the commScripter. In addition the DD source files are contained. The FF DD source files are grouped in a list of common files that can be used for all devices and a list of DD sources files specific for the Modbus device. See chapter commScripter Start HART (113). The output files generated by the FF Tokenizer are also provided as well as the CFF file that is created by commScripter.

5.4.1 FF DD files

For the common DD input files click here (113).

Device-specific DD input files:

The following files are specific for the Modbus device:

- Tools\DD\cs_modb_0309\cs_modbus.ddl
- Tools\DD\cs_modb_0309\cs_modbus.h
- Tools\DD\cs_modb_0309\cs_modbus_fblks.ddl
- Tools\DD\cs_modb_0309\cs_modbus_resb.ddl
- Tools\DD\cs_modb_0309\cs_modbus_resb.h
- Tools\DD\cs_modb_0309\cs_modbus_tblk.ddl
- Tools\DD\cs_modb_0309\cs_modbus_tblk.h
- Tools\DD\cs_modb_0309\tok_cs_modbus.bat

5.4.2 FF DD/CFF files

The following files are generated by the FF Tokenizer based on the input files described in the section DD files above:

- Tools\DD\0309\0601.ff5 DD binary file (V5 format)
- Tools\DD\0309\0601.sy5 DD symbol file (V5 format)
- Tools\DD\0309\0601.ffo DD binary file (V4 legacy format)
- Tools\DD\0309\0601.sym DD symbol file (V4 legacy format)
- Tools\DD\0309\symbols.txt Ids of DD symbols

The CFF (FF common file format) file is generated by the commScripter and provides an offline description of the device blocks and parameters

- Tools\DD\0309\060101.cff

5.4.3 EDD/GSD files

The following EDD files are specific for the Modbus device:

- Tools\EDD\0309\Softing_Sample_0309.devices
- Tools\EDD\0309\0117_0309_01.ddl
- Tools\EDD\0309\0117_0309_01_macros.h
- Tools\EDD\0309\0117_0309_01_menus.inc
- Tools\EDD\0309\softing_macros.h
- Tools\EDD\0309\phy_std.inc
- Tools\EDD\0309\phy_ext_acme.inc
- Tools\EDD\0309\funcX_AI.inc
- Tools\EDD\0309\funcX_AO.inc
- Tools\EDD\0309\funcX_DI.inc
- Tools\EDD\0309\funcX_DO.inc
- Tools\EDD\0309\transX_std.inc
- Tools\EDD\0309\transX_diag.inc
- Tools\EDD\0309\trans1_ext_mult.inc
- Tools\EDD\0309\pa_status.h
- Tools\EDD\0309\pa_units.h
- Tools\EDD\0309\prefix.h
- Tools\EDD\0309\rsp_codes.h
- Tools\EDD\0309\standard.dct

The GSD file, generated by commScripter, contains the characteristic communication features of the HART device.

- Tools\GSD\0309\SOFT3809.gsd

5.4.4 commScripts

The following script files need to be processed in the commScripter tool with the DD binary files generated by the FF Tokenizer:

- Tools\commScripts\MODBUS_0309\cs-modbus.cct (main script file)
- Tools\commScripts\MODBUS_0309\cs-modbus-mapping.cct
- Tools\commScripts\MODBUS_0309\cs-modbus-transducer.cct
- Tools\commScripts\MODBUS_0309\cs-modbus-values.cct

The following batch file can be used to invoke the commScripter and create flashable SREC file and CFF file for the Modbus device:

- Tools\commScripts\Cr8_Modbus_output.bat
- Tools\commScripts\Cr8_PA_Modbus_output.bat

5.5 Batch generation

You can use batch files to invoke the commScripter. For every example device a generation batch is available. The batch files are located in subdirectory *Tools\commScripts*:

- Cr8_<device>_output.bat
- Cr8_PA_<device>.bat
- device = HART, MODBUS, STH, STM

5.6 Renesas Flash Programmer

Note

For details on how to flash a commScript configuration database see Chapter 4.7 107.

In the subdirectory Tools\RFP you will find these three files:

- RFP\commScripterData\commScripterFFDatabase.rpj (Project file for Renesas Flash Programmer for flashing FF configuration table))
- RFP\commScripterData\commScripterPADatabase.rpj (Project file for Renesas Flash Programmer for flashing PA configuration table)
- RFP\RX64M Group.fcf: Description of RX64M processor

Addititional RFP project files are available to flash the FF/PA commDevice firmware.

- RFP\Firmware\commModuleFFFirmware.rpj: project file for flashing FF firmware
- RFP\Firmware\commModulePAFirmware.rpj: project file for flashing PA firmware
- RFP\Firmware\commModuleCompleteFirmware.rpj: project file for restoring the complete commModule firmware (e.g. after a failed firmware update)
- RFP\Firmware\RX64M Group.fcf: Description of RX64M processor.

Firmware files are located at:

- *Firmware\FF_StartHART*\fbk3_sth_ff_r.mot
- Firmware\PA_StartHART\fbk3_sth_pa_r.mot
- Firmware\Complete\fbk3_sth_ff_dld_pa_dld_r.mot

In addition subdirectory Tools\RFP_E2Lite will contain the same project files which allow to use a Renesas E2 Lite instead of Renesas E1.

5.6.1 Firmware download tools

To support firmware update via fieldbus and generation of downloadable files, the commKit package contains the deviceUpdater tools and the GenDomain tool (see Chapter Firmware download 124). You will find the GenDomain tool at *Tools\GenDomain\GenDomain.exe.*

The deviceUpdater tool comes in 2 variants, one for FF and one for PA. Installers can be found at *Tools\deviceUpdater-FF* and *Tools\deviceUpdater-PA*.

For more details see the user guide deviceUpdater-PA and deviceUpdater-FF.

6 Firmware download

As of version 3.60, the commKit package supports a firmware update via fieldbus. For FF, a standard software download protocol is implemented for class 3 FF devices using single domain download. Any host system providing software download capability can be used.

Additionally, a Softing proprietary extended mode is available which allows for individual downloads of the commModule firmware and the commScripter mapping table data. The extended mode can only be executed using with the Softing deviceUpdater tool. It is also available for Profibus PA (which does not specify a standard firmware update protocol at all).

CAUTION

Due to power limitations, a firmware update is only permitted for devices which draw more than 16mA from the fieldbus link (as set in the DEVICE_SETTINGS section of the commScript). At lower current levels the device may fail to write the new program code into the internal flash memory.

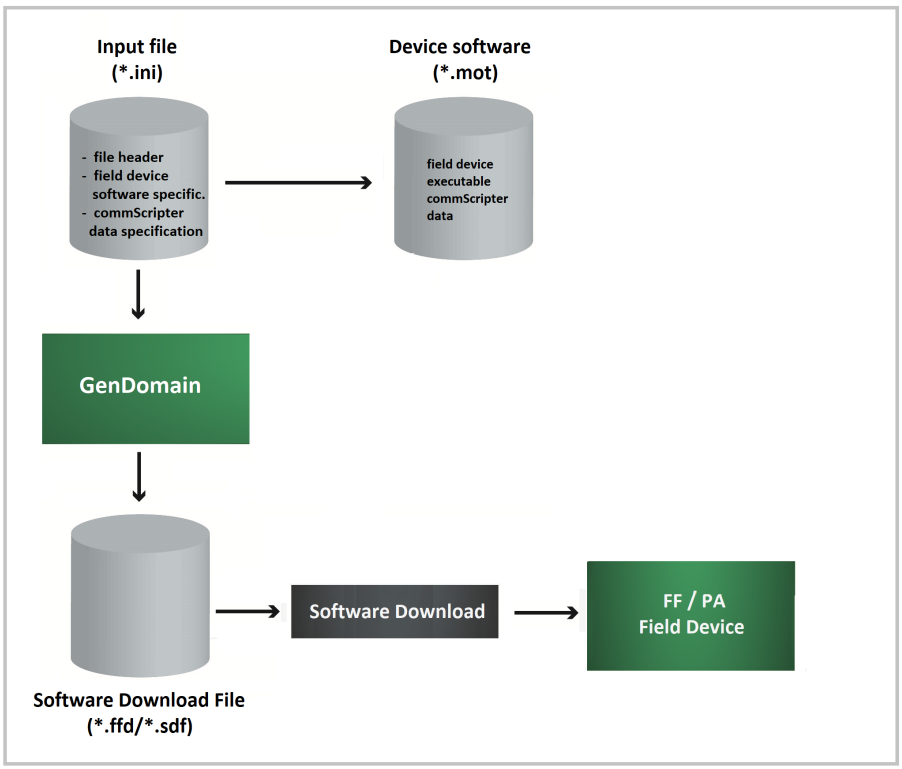
Softing's GenDomain tool can be used to generate a firmware file as a Software Download File. The .mot file can contain an FF/PA Field Device Software, a commScripter data file or both. It is converted into a binary file and compressed by LZSS algorithm. The resulting Software Download File can be downloaded to a field device using Softing's deviceUpdater-FF/deviceUpdater-PA tools which support downloading Field Device Software and commScripter data or individually only the Field Device Software or only the commScripter data.

For PA an extension is not defined by the PA specification. Therefore, update is possible only with deviceUpdater-PA. In the FF specification the software download is specified. Therefore, it is possible to download the combined package Field Device Software and commScripter data by any FF host supporting Software Download.

CAUTION

To allow the standard FF host to perform a valid firmware update, the domain file must always contain the field device software as well as the commScripter data. Otherwise, a firmware update may impact on the functionality of the device. This is also true for PA, even if there are no download tools other than deviceUpdater.

Download concept



With the information from the input file and with the device executable the GenDomain tool generates the Software Download File.

6.1 GenDomain input file

All the necessary information which GenDomain needs for generating the Software Download File are given in a GenDomain input file which has the filename extension .ini.

The information in the input file is different for FF and PA:

FF input file

The following information is necessary for an FF Field Device

- Download file header
  - Version
  - Manufacturer id
  - Device family / device type
  - Device revision
  - DD revision
  - Software revision
  - Software name
  - Domain name

- Field Device Software / commScripter data information
  - Hardware id
  - Filename
  - Startaddress
  - Endaddress
  - Type (Field Device Software or commScripter data)
  - Use compression

The content of the download file header is defined by the Fieldcomm Group in the in the document FF-883 FS 1.3 Software Download Addendum.

The following is an example of an ini-file for an FF Field Device Software and commScripter data:

Example

```
[FF Domain Header]
  Version = 1
  Manufacture ID = 0x1E6D11
  Device Family = "0320"
  Device Type = 0x0320
  DEV_REV = 0x06
  DD_REV = 0x01
  Software Revision = "3-60    "
  Software Name = "FD-SW    "
  Domain Name = "FD-DOM   "

  [Settings]
  PACK1_FW_NAME = .\fbk3_sth_ff_r.mot
  PACK2_FW_NAME = .\cs_output_FF.mot

  [Private Header]
  HARDWARE_ID = "FBK3_RX64M"
  N_PACKETS   = 2
  PACK1_START_ADDR = 0xFFE00000
  PACK1_END_ADDR   = 0xFFEF7FFF
  PACK1_IS_COMPRESSED = TRUE
  PACK1_TYPE = DWNLD_FD_PART
```

```
PACK1_NAME = "FFFirmware V3.60"
PACK2_START_ADDR = 0xFFEF8000
PACK2_END_ADDR   = 0xFFEFFFFF
PACK2_IS_COMPRESSED = FALSE
PACK2_TYPE = DWNLD_CSCR_PART
PACK2_NAME = "StarterKITcS"
```

Keyword	Description
[FF Domain Header]	Name of the section - must not be changed! This section contains header information.
Version	Version of the download header. Fixed by Fieldcomm Group to be 1.
Manufacture ID	The manufacturer id as assigned by Fieldcomm Group, represented as hexadecimal number. Note: This matches the commScript entry FF_DEVICE_IDENT::ManufacturerId
Device Family	The device family as assigned by the device vendor. Is a string with hexadecimal digits. Note: This matches the commScript entry FF_DEVICE_IDENT:: DeviceType
Device Type	The device type of the device as assigned by the device vendor. Is a hexadecimal value with 4 digits. Note: The value should match the device family because currently support of multiple device types within one device family is not supported in the FD-SW Note 1: This matches the commScript entry FF_DEVICE_IDENT:: DeviceType
Dev REV	Revision of the device as assigned by the vendor. Note: This matches the commScript entry FF_DEVICE_IDENT:: DeviceRevision
DD REV	Device Description revision used for the DD of the device.
Software Name	Name of the software
Domain Name	Name of the domain that is used for Software Download
[Settings]	Name of the section - must not be changed! This section contains the file names of the .mot files for all packages. The number of entries has to match the value assigned to N_PACKETS.
PACK<i>_FW_NAME	The defines the file name of the file which should be added to the download data which will be downloaded to the PA device. It should be in Motorola format. i is 1, …, N_PACKETS

[Private Header]	Name of the section - must not be changed! This section contains the file names of the .mot files for all packages. The number of entries should match the value assigned to N_PACKETS.
HARDWARE_ID	The Hardware Identification is used to ensure the consistence between hardware platform and the software to be downloaded. A field device accepts a download file only if its own hardware identification is identical to the hardware identification specified in the download file. The Hardware Identification is a vendor-specific visible string. Valid characters are all upper and lower case letters A - Z, the numbers 0 - 9, and the hyphen "-" character. All other characters are invalid. The length of the Device ID string shall be exactly 10 characters.
N_PACKETS	The number of packages that will be contained in the download data. Currently only the values 1 or 2 are supported. N_PACKETS = 2 will contain the Field Device Software and the commScripter data. N_PACKETS = 1: In this case either Field Device Software or commScripter data will be contained
PACK<i>_START_ADDR	This defines the start address of the area to be used in the Field Device Software or commScripter data. The values have to be: 0xFFE00000 (for FF Field Device Software) 0xFFEF8000 (for FF commScripter data) Note: These address information must not be changed. i is 1, …, N_PACKETS
PACK<i>_END_ADDR	This defines the end address of the area to be used in the Field Device Software or commScripter data. The values have to be: 0xFFEF7FFF (for FF Field Device Software) 0xFFEFFFFF (for FF commScripter data) Note: These address information must not be changed. i is 1, …, N_PACKETS
PACK<i>_IS_COMPRESSED	This field defined if the data will be compressed or not. The possible values are: TRUE: use compression FALSE: data will not be compressed It is recommended to use compression for the firmware part. i is 1, …, N_PACKETS

PA input file

- Download file header
  - Header revision
  - Manufacturer id
  - Device id

- Hardware id
- Software revision

- Field Device Software / commScripter data information
  - Filename
  - Startaddress
  - Endaddress
  - Type (Field Device Software or commScripter data)
  - Use compression

The following shows an ini-file for a PA Field Device Software and commScripter data:

```
[PA Download File Header]
HEADER_REVISION = 16
MANUFACTURER_ID = 279
DEVICE_ID = "CS-START-HART"
SOFTWARE_REVISION = "V3.60.0          "
HARDWARE_ID = "FBK3_RX64M"
N_PACKETS   = 2
PACK1_START_ADDR = 0xFFF00000
PACK1_END_ADDR   = 0xFFFD7FFF
PACK1_IS_COMPRESSED = TRUE
PACK1_TYPE = DWNLD_FD_PART
PACK1_NAME = "PAFirmware V3.60"
PACK2_START_ADDR = 0xFFFD8000
PACK2_END_ADDR   = 0xFFFDFFFF
PACK2_IS_COMPRESSED = FALSE
PACK2_TYPE = DWNLD_CSCR_PART
PACK2_NAME = "StarterKITcS"

[Settings]
PACK1_FW_NAME = .\fbk3_sth_pa_r.mot
PACK2_FW_NAME = .\cs_output_PA.mot
```

Keyword	Description
[PA Download File Header]	Name of the section - must not be changed! This section contains header information and the definitions of the contained packages. The number of entries must match the value assigned to N_PACKETS
HEADER_REVISION	The version number for this header which defines the format of the download data. The value is 16. Other format versions are currently not supported.
MANUFACTURER_ID	The MANUFAC_ID as assigned by PNO, represented as decimal or hexadecimal number.

Keyword	Description
DEVICE_ID	The Device ID is an identification code of the field device or sensor/actuator software. Device ID is a vendor-specific visible string. Valid characters are all upper and lower case letters A - Z, the numbers 0 - 9, and the hyphen "-" character. All other characters are invalid. The length of the Device ID string shall be exactly 16 characters.
SOFTWARE_REVISION	Software Revision is a vendor-specific visible string. Valid characters are all upper and lower case letters A - Z, the numbers 0 - 9, and the hyphen "-" character. All other characters are invalid. The length of the Device ID string shall be exactly 16 characters.
HARDWARE_ID	The Hardware Identification is used to ensure the consistence between hardware platform and the software to be downloaded. A field device accepts a download file only if its own hardware identification is identical to the hardware identification specified in the download file. The Hardware Identification is a vendor-specific visible string. Valid characters are all upper and lower case letters A - Z, the numbers 0 - 9, and the hyphen "-" character. All other characters are invalid. The length of the Device ID string shall be exactly 10 characters.
N_PACKETS	The number of packages that will be contained in the download data. Currently only the values 1 or 2 are supported. N_PACKETS = 2 will contain the Field Device Software and the commScripter data. N_PACKETS = 1: In this case either Field Device Software or commScripter data will be contained
PACK<i>_START_ADDR	This defines the start address of the area to be used in the Field Device Software or commScripter data. The values have to be: 0xFFF00000 (for PA Field Device Software) 0xFFFD8000 (for PA commScripter data) Note: These address information must not be changed. i is 1, …, N_PACKETS
PACK<i>_END_ADDR	This defines the end address of the area to be used in the Field Device Software or commScripter data. The values have to be: 0xFFFD7FFF (for PA Field Device Software) 0xFFFDFFFF (for PA commScripter data) Note: These address information must not be changed. i is 1, …, N_PACKETS
PACK<i>_IS_COMPRESSED	This field defined if the data will be compressed or not. The possible values are: TRUE: use compression FALSE: data will not be compressed It is recommended to use compression for the firmware part, i is 1, …, N_PACKETS
PACK<i>_TYPE	1: package contains the firmware

Keyword	Description
	2: package contains commScript data i is 1, …, N_PACKETS
[Settings]	Name of the section - must not be changed! This section contains the file names of the .mot files for all packages. The number of entries has to match the value assigned to N_PACKETS.
PACK<i>_FW_NAME	The defines the file name of the file which should be added to the download data which will be downloaded to the PA device. It should be in Motorola format. i is 1, …, N_PACKETS

6.2 Invocation of GenDomain

GenDomain is a Win32 console application. The required parameters have to be given in the command line. GenDomain does not require a special installation procedure. It is created with Visual Studio 2019. If not already present, the Visual Studio 2019 runtime environment has to be installed.

Command Line:

```
GenDomain.exe input_file [options]
```

Options:	Description:
`-O output_file_path`	Path to the directory where the generated Software Download File should be saved.
`-PACK<x>_PATH mot_file`	Path to the firmware or commScripter data .mot file

Invocation example:

```
GenDomain .\Subfolder\FBK3DOM_sth.ini
```

```
GenDomain .\Subfolder\FBK3DOM_sth.ini -PACK1_PATH myFirmwareFile.mot
```

7 Appendix

7.1 HART to FF/PA data type mapping

HART	FF/PA	Note
BIT_STRING(1)	OCTET_STRING(1)	
BIT_STRING(2)	OCTET_STRING(2)	
BIT_STRING(4)	OCTET_STRING(4)	
DATE		Must be mapped to three consecutive USIGN8 record or array elements
TIME	UNSIGNED32	
ENUM(1)	UNSIGNED8	DD: ENUMERATED (1)
ENUM(2)	UNSIGNED16	DD: ENUMERATED (2)
FLOAT	FLOAT	
LATIN(2)	OCTET_STRING(2)	Allowed range: 2 – 122 (readonly), 119 (writeable)
LATIN(122)	OCTET_STRING(122)	
PACKED(3)	VISIBLE_STRING(4)	
PACKED(6)	VISIBLE_STRING(8)	Len (PACKED) = 3 * (Len (VISIBLE_STRING) / 4). Allowed range for VISIBLE_STRING: 4, 8, … 120 (Len (VISIBLE_STRING) % 4 == 0). For writeable objects the maximum length is 116.
PACKED(90)	VISIBLE_STRING(122)	122 (readonly), 119 (writeable)
UNSIGNED5	UNSIGNED8	The most significant five bits are mapped to the least significant five bits of UNSIGNED8.
UNSIGNED8	UNSIGNED8	
UNSIGNED16	UNSIGNED16	
UNSIGNED24	UNSIGNED32	The most significant byte is skipped
UNSIGNED32	UNSIGNED32	
INTEGER8	INTEGER8	
INTEGER16	INTEGER16	
INTEGER24	INTEGER32	
INTEGER32	INTEGER32	
UNIT	UNSIGNED16	DD: ENUMERATED (2) Unit codes are mapped from FF/PA to HART and vice versa

7.2 Standard PA data structures

The following three standard PA data structures can be used:

Value & Status - Floating Point Structure
This data structure consists of the value and status of floating point parameters.

Data Type: **Value & Status - Floating Point**
Key Attribute: **Index = 101**
commScripter Name: **DS_FLOAT_S**

E	Element Name	Data Type (Index)	Size
1	Value	Float - (8)	4
2	Status	Unsigned8 - (5)	1

Value & Status - Discrete Structure

This data structure consists of the value and status of discrete value parameters.

Data Type: **Value & Status - Discrete**
Key Attribute: **Index = 102**
commScripter Name: **DS_DISC_S**

E	Element Name	Data Type (Index)	Size
1	Value	Unsigned8 - (5)	1
2	Status	Unsigned8 - (5)	1

Scaling Structure
This data structure consists of the static data used to scale floating point values.

Data Type: **Scaling**
Key Attribute: **Index = 36**
commScripter Name: **DS_SCALE**

E	Element Name	Data Type (Index)	Size
1	EU at 100%	Float - (8)	4
2	EU at 0%	Float - (8)	4
3	Units Index	Unsigned16 - (6)	2
4	Decimal Point	Integer8 - (2)	1

7.3 Standard FF data structure

The following three standard FF data structures can be used:

Value & Status - Floating Point Structure

This data structure consists of the value and status of floating point parameters.

Data Type: **Value & Status - Floating Point**
Key Attribute: **Index = 65**
commScripter Name: **DS_FLOAT_S**

E	Element Name	Data Type (Index)	Size
1	Status	Unsigned8 - (5)	1
2	Value	Float - (8)	4

Value & Status - Discrete Structure

This data structure consists of the value and status of discrete value parameters.

Data Type: **Value & Status - Discrete**
Key Attribute: **Index = 66**
commScripter Name: **DS_DISC_S**

E	Element Name	Data Type (Index)	Size
1	Status	Unsigned8 - (5)	1
2	Value	Unsigned8 - (5)	1

Scaling Structure

This data structure consists of the static data used to scale floating point values.

Data Type: **Scaling**
Key Attribute: **Index = 68**
commScripter Name: **DS_SCALE**

E	Element Name	Data Type (Index)	Size
1	EU at 100%	Float - (8)	4
2	EU at 0%	Float - (8)	4
3	Units Index	Unsigned16	2
4	Decimal Point	Integer8	1

7.4 FF resource block parameters

The following table shows the list of standard parameters used by the commScript resource block. Device-specific changes can be added at the end of the default parameter list.

Relative Index	Parameter
0	RESOURCE_BLOCK_2 (block header)
1	ST_REV
2	TAG_DESC
3	STRATEGY
4	ALERT_KEY
5	MODE_BLK
6	BLOCK_ERR
7	RS_STATE
8	TEST_RW
9	DD_RESOURCE
10	MANUFAC_ID
11	DEV_TYPE
12	DEV_REV
13	DD_REV
14	GRANT_DENY
15	HARD_TYPES
16	RESTART
17	FEATURES
18	FEATURE_SEL
19	CYCLE_TYPE
20	CYCLE_SEL
21	MIN_CYCLE_T
22	MEMORY_SIZE
23	NV_CYCLE_T
24	FREE_SPACE
25	FREE_TIME
26	SHED_RCAS
27	SHED_ROUT
28	FAULT_STATE
29	SET_FSTATE
30	CLR_FSTATE
31	MAX_NOTIFY
32	LIM_NOTIFY
33	CONFIRM_TIME
34	WRITE_LOCK
35	UPDATE_EVT
36	BLOCK_ALM
37	ALARM_SUM
38	ACK_OPTION
39	WRITE_PRI
40	WRITE_ALM

Relative Index	Parameter
41	ITK_VER
42	SOFTWARE_REV
43	HARDWARE_REV
44	CS_SCRIPT_REV
45	CS_CONTENT_REV
46	APPL_TUNNEL_STATE
47	APPL_TUNNEL_REQ
48	APPL_TUNNEL_RES
49	FD_VER
50	FD_FAIL_ACTIVE
51	FD_OFFSPEC_ACTIVE
52	FD_MAINT_ACTIVE
53	FD_CHECK_ACTIVE
54	FD_FAIL_MAP
55	FD_OFFSPEC_MAP
56	FD_MAINT_MAP
57	FD_CHECK_MAP
58	FD_FAIL_MASK
59	FD_OFFSPEC_MASK
60	FD_MAINT_MASK
61	FD_CHECK_MASK
62	FD_FAIL_ALM
63	FD_OFFSPEC_ALM
64	FD_MAINT_ALM
65	FD_CHECK_ALM
66	FD_FAIL_PRI
67	FD_OFFSPEC_PRI
68	FD_MAINT_PRI
69	FD_CHECK_PRI
70	FD_SIMULATE
71	FD_RECOMMEN_ACT

7.4.1 Resource block views

VIEW_1

Relative Index	Parameter	Parameter length
1	ST_REV	2
5	MODE_BLK	4
6	BLOCK_ERR	2
7	RS_STATE	1
25	FREE_TIME	4
28	FAULT_STATE	1
37	ALARM_SUM	8
50	FD_FAIL_ACTIVE	4
51	FD_OFFSPEC_ACTIVE	4
52	FD_MAINT_ACTIVE	4
53	FD_CHECK_ACTIVE	4
71	FD_RECOMMEN_ACT	2
		40

VIEW_2

Relative Index	Parameter	Parameter length
1	ST_REV	2
14	GRANT_DENY	2
18	FEATURE_SEL	2
20	CYCLE_SEL	2
23	NV_CYCLE_T	4
24	FREE_SPACE	4
26	SHED_RCAS	4
27	SHED_ROUT	4
32	LIM_NOTIFY	1
33	CONFIRM_TIME	4
34	WRITE_LOCK	1
		30

VIEW_3

Relative Index	Parameter	Parameter length
1	ST_REV	2
5	MODE_BLK	4
6	BLOCK_ERR	2
7	RS_STATE	1
25	FREE_TIME	4
28	FAULT_STATE	1
37	ALARM_SUM	8

Relative Index	Parameter	Parameter length
50	FD_FAIL_ACTIVE	4
51	FD_OFFSPEC_ACTIVE	4
52	FD_MAINT_ACTIVE	4
53	FD_CHECK_ACTIVE	4
71	FD_RECOMMEN_ACT	2
		49

VIEW_4

Relative Index	Parameter	Parameter length
1	ST_REV	2
3	STRATEGY	2
4	ALERT_KEY	1
10	MANUFAC_ID	4
11	DEV_TYPE	2
12	DEV_REV	1
13	DD_REV	1
15	HARD_TYPES	2
17	FEATURES	2
19	CYCLE_TYPE	2
21	MIN_CYCLE_T	4
22	MEMORY_SIZE	2
31	MAX_NOTIFY	1
38	ACK_OPTION	2
39	WRITE_PRI	1
41	ITK_VER	2
49	FD_VER	2
54	FD_FAIL_MAP	4
55	FD_OFFSPEC_MAP	4
56	FD_MAINT_MAP	4
57	FD_CHECK_MAP	4
58	FD_FAIL_MASK	4
59	FD_OFFSPEC_MASK	4
60	FD_MAINT_MASK	4
61	FD_CHECK_MASK	4
66	FD_FAIL_PRI	1
67	FD_OFFSPEC_PRI	1
68	FD_MAINT_PRI	1
69	FD_CHECK_PRI	1
		69

7.5 FF transducer block parameters

The following list shows the standard parameters of the commDevice transducer blocks.

Relative Index	Parameter
0	Block Header
1	ST_REV
2	TAG_DESC
3	STRATEGY
4	ALERT_KEY
5	MODE_BLK
6	BLOCK_ERR
7	UPDATE_EVT
8	BLOCK_ALM
9	TRANSDUCER_DIRECTORY
10	TRANSDUCER_TYPE
11	TRANSDUCER_TYPE_VER
12	XD_ERROR
13	COLLECTION_DIRECTORY

7.5.1 Transducer block views

VIEW_1

Relative Index	Parameter	Parameter length
1	ST_REV	2
5	MODE_BLK	4
6	BLOCK_ERR	2
10	TRANSDUCER_TYPE	2
11	TRANSDUCER_TYPE_VER	2
12	XD_ERROR	1
		13

VIEW_2

Relative Index	Parameter	Parameter length
1	ST_REV	2
10	TRANSDUCER_TYPE	2
11	TRANSDUCER_TYPE_VER	2
		6

VIEW_3

Relative Index	Parameter	Parameter length
1	ST_REV	2
5	MODE_BLK	4
6	BLOCK_ERR	2
10	TRANSDUCER_TYPE	2
11	TRANSDUCER_TYPE_VER	2
12	XD_ERROR	1
		13

VIEW_4

Relative Index	Parameter	Parameter length
1	ST_REV	2
6	STRATEGY	2
7	ALERT_KEY	1
10	TRANSDUCER_TYPE	2
11	TRANSDUCER_TYPE_VER	2
		9

7.6 Resource block errors

The following resource block errors are available:

- RESB_ERR_MEM_FAIL
- RESB_ERR_STATIC_DATA_LOST
- RESB_ERR_NV_DATA_LOST
- RESB_ERR_NEEDS_MAINT_SOON
- RESB_ERR_NEEDS_MAINT_NOW

7.7 FF sub-status of process values

For UNCERTAIN and BAD quality FF sub-status values provide additional information. The following subset of the FF sub-status is available:

- BAD_DEV_FAIL
- BAD_SENSOR_FAIL
- UNC_SUBST_VAL
- UNC_SENSOR_CONF_NOT_ACC
- UNC_SUB_NORMAL
- UNC_EU_RANGE_VIOLATION

7.8 FF transducer block errors

The following transducer block error codes are available:

- TB_ERR_BLOCK_CFG
- TB_ERR_LOC_OVERRIDE
- TB_ERR_SENSOR_FAIL
- TB_ERR_OUTPUT_FAIL
- TB_ERR_READBACK_FAIL

In case of TB_ERR_BLOCK_CFG the transducer block switches to OOS and the status of the PV and readback parameters switch to BAD|OOS.

7.9 PROFIBUS PA – Profile Ident Numbers

Profile-specific ident numbers supported by commDevices.

Device Type	Ident Number
Transmitter 1 AI	0x9700
Transmitter 2 AI	0x9701
Transmitter 3 AI	0x9702
Transmitter 4 AI	0x9703
Transmitter 5 AI	0x9704
Transmitter 7 AI	0x9706
Transmitter 8 AI	0x9707
Transmitter 15 AI	0x970E
Actuator 1 AO	0x9710
Discrete Input 1 DI	0x9720
Discrete Output 1 DO	0x9730
Discrete Output 4 DO	0x9733

7.10 PROFIBUS PA physical block parameters

The following table shows the list of default parameters used by the commScript physical block. Device-specific parameters can be added at the end of the default parameter list.

Relative Index	Parameter
0	BLOCK_OBJECT
1	ST_REV
2	TAG_DESC
3	STRATEGY
4	ALERT_KEY
5	TARGET_MODE
6	MODE_BLK
7	ALARM_SUM
8	SOFTWARE_REVISION
9	HARDWARE_REVISION
10	DEVICE_MAN_ID
11	DEVICE_ID
12	DEVICE_SER_NUM
13	DIAGNOSIS
14	DIAGNOSIS_EXT
15	DIAGNOSIS_MASK
16	DIAGNOSIS_MASK_EXT
17	DEVICE_CERTIFICATION
18	WRITE_LOCKING
19	FACTORY_RESET
20	DESCRIPTOR
21	DEVICE_MESSAGE
22	DEVICE_INSTAL_DATE
23	NULL_PARAM
24	IDENT_NUMBER_SELECTOR
25	HW_WRITE_PROTECTION
26	FEATURE
27	COND_STATUS_DIAG
28	DIAG_EVENT_SWITCH
29	NULL_PARAM
30	NULL_PARAM

Relative Index	Parameter
31	NULL_PARAM
32	NULL_PARAM
33	CS_SCRIPT_REV
34	CS_CONTENT_REV
35	APPL_TUNNEL_STATE
36	APPL_TUNNEL_REQ
37	APPL_TUNNEL_RES

7.10.1 PA physical block views

VIEW_1

Relative Index	Parameter	Parameter length
1	ST_REV	2
6	MODE_BLK	3
7	ALARM_SUM	8
13	DIAGNOSIS	4
		17

VIEW_2

Relative Index	Parameter	Parameter length
1	ST_REV	2
6	MODE_BLK	3
7	ALARM_SUM	8
13	DIAGNOSIS	4
14	DIAGNOSIS_EXT	6
18	WRITE_LOCKING	2
25	HW_WRITE_PROTECTION	1
26	FEATURE	8
27	COND_STATUS_DIAG	1
		40

7.11 PROFIBUS PA transducer block parameters

The following list shows the standard parameters of a PA transducer block created by the commScripter firmware.

Relative Index	Parameter
0	BLOCK_OBJECT
1	ST_REV
2	TAG_DESC
3	STRATEGY
4	ALERT_KEY
5	TARGET_MODE
6	MODE_BLK
7	ALARM_SUM

7.11.1 PA transducer block views

VIEW_1

Relative Index	Parameter	Parameter length
1	ST_REV	2
6	MODE_BLK	3
7	ALARM_SUM	8
		13

7.12 PROFIBUS PA diagnostics

Classic status: In case of classic status the commScripter setting for the diagnostic condition determine to which process value status and DIAGNOSIS value the condition is mapped. The priority is according to the PROFIBUS PA specification V3.02 as described in table 79. When two or more conditions are active at the same time the process value status is set according to the condition with the highest priority. All bits in DIAGNOSIS value that are related to the active conditions will be set in this case.

Condensed Status: In case of condensed status the mapping of condition to process value status and DIAGNOSIS bit is controlled by the Diag_Event_Switch.Link_Status_Array. The mapping defined in the commScripter diagnosis mapping will provide default values that are used for the Diag_Event_Switch.Link_Status_Array. Link_Status_Array is an array of 48 conditions. The values will be set for the conditions 0 - 47 as defined in the commScript. If a condition is not used in the commScript the related byte in Link_Status_Array will be set to 0.

By writing to Diag_Event_Switch.Link_Status_Array it is possible to define a different mapping for process value status and DIAGNOSIS bit. All permitted values for Link_Status_Array can be written. If a condition is not used it is still possible to write any permitted value but the value is not updated. It will still be 0.

Permitted values for Link_Status_Array are defined in PROFIBUS PA specification V3.02 Table 44. The definition is as follows:

Status code (low nibble = Bit 0 - 3):
0: Diagnostic event has no affect on the status.
Status will be **GOOD . ok**

1: Diagnostic event is treated as **maintenance request**.
Status will be **GOOD . maintenance required**.

2: Diagnostic event is treated as **immediate maintenance request**.
Status will be **GOOD . maintenance demanded**

3: Diagnostic event is treated as **immediate maintenance request**.
Status will be **UNCERTAIN . maintenance demanded**

4: Diagnostic event is treated as **failure**.
Status will be **BAD . maintenance alarm**

5: Diagnostic event is treated as **invalid process condition**.
Value is conditionally usable.
Status will be **UNCERTAIN . process related, no maintenance**

6: Diagnostic event is treated as invalid process condition.
Value is not usable.
Status will be **BAD . process related, no maintenance**

7: Diagnostic event is treated as **function check without usable value**.
Status will be **BAD . function check** / local override

8: Diagnostic event is treated as **function check with usable value**.
Status will be **GOOD . function check**.

9 .. 15 reserved
Diagnosis (high nibble = Bit 4 - 7):

0: Diagnostic event has no affect on the diagnosis.
DIAGNOSIS: No additional bit will be set.

1: Diagnostic event is treated as **maintenance request**.
DIAGNOSIS: **DIA_MAINTENANCE** will be set.

2: Diagnostic event is treated as **immediate maintenance request**.
DIAGNOSIS: **DIA_MAINTENANCE_DEMAND** will be set.

3: Diagnostic event is treated as **failure**.
DIAGNOSIS: **DIA_MAINTENANCE_ALARM** will be set.

4: Diagnostic event is treated as **invalid process condition**.
DIAGNOSIS: **DIA_INV_PRO_COND** will be set.

5: Diagnostic event is treated as function check or simulation.
DIAGNOSIS: **DIA_FUNCTION_CHECK** will be set.

6 .. 15: reserved

Using a reserved value for high or low nibble is not permitted.

The attributes Diag_Event_Switch.Slot and Diag_Event_Switch.Index are not used. They are set to value = 0. It is possible to write any value to these attributes but the value would remain 0.

7.13 Softing demo transducer block parameters and functions

The Softing Demo TB provides channels to source and sink process values to input function blocks (AI, DI) from output function blocks (AO, DO). Process values generated within the Demo TB can be configured in some properties and serve to provide predictable values with good status to the function blocks. To be used in conjunction with analog and discrete function blocks, the process values and their according configuration parameters are available in analog (= FLOAT) as well as in discrete (= UNSIGNED8) data types.

The DemoTB is deactivated automatically as soon as the Manufacturer ID or Device Type is changed. However, for clearness of implementation the according code referring the Demo-TB should be removed from the commScript in any manufacturer adapted version.

Note

For more information on the internal function of the Demo TB, please refer to the user guide *Field Device Software* version 3.60.

7.14 commScripter command line options

Options	Notes
-device=	Four device variants are supported: ▪ FF+HART ▪ FF+MODBUS ▪ PA+HART ▪ PA+MODBUS Please note that for each device variant a separate commScripter license is required.
-script=	Path and name of the (main-) commScript file (.cct file).
-sym_file=	Path and name of the symbol file (.sy5 file). For FF only
-output_path=	Directory where the commScripter stores the commScript configuration table (cs_output_FF.mot or cs_output_PA.mot).
-cff_path=	Directory where the commScripter stores the CCF file. If no cff_path is specified the commScripter sets cff_path=output_path. For FF only
-gsd_path=	Directory where the commScripter stores the GSD file. If no gsd_path is specified the commScripter does not generate a GSD file. For PROFIBUS PA only
-gsd_vendor=	Four character short name of the device vendor (e.g. SOFT, ACME, …) The short name becomes part of the GSD file name (e.g. gsd_vendor: SOFT, manufacturer profile number: 0x3807SOFT3807.GSD). If no gsd_vendor is specified the commScripter sets gsd_vendor=SOFT. For PROFIBUS PA only
-testmode	When testmode is activated the commScripter checks the input files for correctness but does not generate any output files. testmode can be used when no commScripter license is available.

7.15 commScript unit codes

The following chapters describe the engineering units that are supported by commScripter MBP. For FF to HART mapping the standard mapping is used as defined in FOUNDATION™ Technical Note 21 Engineering Units Mapping from HART® to FOUNDATION™ Fieldbus, TN-021, 1.0.

The following tables might have a row for FF unit and PA unit if some unit values differ. PA unit is listed only when it is different from the FF unit.

7.15.1 Percent

Keyword	Description	FF/PA Unit	HART Unit
PERCENT	Percent of full range	1342	57

7.15.2 Temperature

Keyword	Description	FF/PA Unit	HART Unit
CELSIUS	Degrees Celsius	1001	32
FAHRENHEIT	Degrees Fahrenheit	1002	33
RANKINE	Degrees Rankine	1003	34
KELVIN	Kelvin	1000	35

7.15.3 Pressure

Keyword	Description	FF/PA Unit	HART Unit
inH20_68F	inches of water at 68 degrees Fahrenheit	1148	1
inHg_0C	inches of mercury at 0 degrees Celsius	1156	2
ftH20_68F	feet of water at 68 degrees Fahrenheit	1154	3
mmH20_68F	millimeters of water at 68 degrees Fahrenheit	1151	4
mmHg_0C	millimeters of mercury at 0 degrees Celsius	1158	5
PSI	pounds per square inch unreferenced or differential pressure	1141	6
BAR		1137	7
MILLIBAR		1138	8
GSCM	grams per square centimeter	1144	9
KSCM	kilograms per square centimeter	1145	10
PASCAL		1130	11
KILOPASCAL		1133	12
TORR		1139	13
ATMOSPHERE		1140	14
MEGAPASCAL		1132	237
inH20_4C	inches of water at 4 degrees Celsius	1147	238
mmH20_4C	millimeters of water at 4 degrees Celsius	1150	239

7.15.4 Flow

Volume flow unit codes

Keyword	Description	FF Unit	PA Unit	HART Unit
CFM	cubic feet per minute	1357		15
gal/m GPM	US gallons per minute	1363		16
L/min	liters per minute	1352		17
ImpGal/min	imperial gallons per minute	1368		18
Cm/h	cubic meters per hour	1349		19
gal/s	US gallons per second	1362		22
Mgal/d	million US gallons per day	1366		23
L/s	liters per second	1351		24
ML/d	million liters per day	1355		25
CFS	cubic feet per second	1356		26
CFD CF/d	cubic feet per day	1359		27
Cm/s	cubic meters per second	1347		28
Cm/d	cubic meters per day	1350		29
ImpGal/h	imperial gallons per hour	1369		30
ImpGal/d	imperial gallons per day	1370		31
NCm/h	normal cubic meters per hour (0°C, 1atm)	1524	1590	121
NL/h	normal liters per hour (0°C, 1atm)	1534	1594	122
SCFM	standard cubic feet per minute	1360		123
CFH	cubic feet per hour	1358		130
Cm/min	cubic meter per minute	1348		131
bbl/s	barrels per second	1371		132
bbl/min	barrels per minute	1372		133
bbl/h	barrels per hour	1373		134
bbl/d	barrels per day	1374		135
gal/h	US gallons per hour	1364		136
ImpGal/s	imperial gallons per second	1367		137
L/h	liters per hour	1353		138
SL/s	standard liters per second (20°C, 1atm)	1537	1600	180
Nm3/d	normal cubic meters per day (0°C, 1atm)	1525	1591	181
Nm3/min	normal cubic meters per minute (0°C, 1atm)	1523	1589	182
Nm3/s	normal cubic meters per second (0°C, 1atm)	1522	1588	183
SCFH	standard cubic feet per hour (32°F, 1atm)	1361		185
Sm3/d	standard cubic meters per day (20°C, 1atm)	1530	1599	187

Keyword	Description	FF Unit	PA Unit	HART Unit
Sm3/h	standard cubic meters per hour (20°C, 1atm)	1529	1598	188
Sm3/min	standard cubic meters per minute (20° C,1atm)	1528	1597	189
Sm3/s	standard cubic meters per second (20° C,1atm)	1527	1596	190
gal/d	US gallons per day	1365		235

Mass flow unit codes

Keyword	Description	FF/PA Unit	HART Unit
g/s	grams per second	1318	70
g/min	grams per minute	1319	71
g/h	grams per hour	1320	72
kg/s	kilograms per second	1322	73
kg/min	kilograms per minute	1323	74
kg/h	kilograms per hour	1324	75
kg/d	kilograms per day	1325	76
t/min	metric tons per minute	1327	77
t/h	metric tons per hour	1328	78
t/d	metric tons per day	1329	79
lb/s	pounds per second	1330	80
lb/min	pounds per minute	1331	81
lb/h	pounds per hour	1332	82
lb/d	pounds per day	1333	83
STon/min	short tons per minute	1335	84
STon/h	short tons per hour	1336	85
STon/d	short tons per day	1337	86
LTon/h	long tons per hour	1340	87
LTon/d	long tons per day	1341	88

7.15.5 Volume

Keyword	Description	FF Unit	PA Unit	HART Unit
BARREL		1051		46
Bbl	liquid barrels	1052		124
BUSHEL		1050		110
CFD (/CF/d)	cubic feet per day (the former unit CF/d is also supported)	1359		27
Cft	cubic feet	1043		112
Cin	cubic inches	1042		113
Cm	cubic meters	1034		43
Cyd	cubic yards	1044		111
GALLON	US gallons	1048		40
gal/m (GPM)	US gallon per minute (the former unit GPM is also supported)	1363		16
HECTOLITER		1041		236
ImpGal	imperial gallons	1049		42
LITER		1038		41
NCM	normal cubic meters (0°C, 1atm)	1521	1573	166
NL	normal liters (0°C, 1atm)	1531	1574	167
SCF	standard cubic feet U.S. System	1053		168

7.15.6 Mass

Keyword	Description	FF/PA Unit	HART Unit
GRAM		1089	60
KG	kilograms	1088	61
T	metric ton	1092	62
POUND		1094	63
STon	short tons	1095	64
LTon	long tons	1096	65
OUNCE		1093	125

7.15.7 Mass per Volume

Keyword	Description	FF Unit	PA Unit	HART Unit
SGU	specific gravity unit	1114		90
g/Ccm	grams per cubic centimeter	1100		91
kg/Cm	kilograms per cubic meter	1097		92
plb/gal	pounds per gallon	1108		93
lb/Cft	pounds per cubic foot	1107		94
g/ml	grams per milliliter	1104		95
kg/L	kilograms per liter	1103		96
g/L	grams per liter	1105		97
lb/Cin	pounds per cubic inch	1106		98
STon/Cyd	short tons per cubic yard	1109		99
degTwad	degrees twaddell	1110		100
degBaum	degrees baume heavy	1111		102
degAPI	degrees API	1113		104
ug/L	micrograms per liter	1676	1559	146
ug/Cm	micrograms per cubic meter	1677	N.A.	147

7.15.8 Length

Keyword	Description	FF / PA Unit	HART Unit
FEET		1018	44
METER		1010	45
INCHES		1019	47
CENTIMETER		1012	48
MILLIMETER		1013	49

7.15.9 Time

Keyword	Description	FF / PA Unit	HART Unit
MINUTE		1058	50
SECOND		1054	51
HOUR		1059	52
DAY		1060	53

7.15.10 Velocity

The table contains linear and angular velocity unit codes which cannot be converted into each other.

Keyword	Description	FF / PA Unit	HART Unit
m/s	meter per second	1061	21
f/s	feet per second	1067	20
in/s	inches per second	1066	114
in/min	inches per minute	1069	115
ft/min	feet per minute	1070	116
m/h	meter per hour	1063	120
rev/s	revolutions per second	1084	118
RPM	revolutions per minute	1085	119

7.15.11 Energy

Keyword	Description	FF / PA Unit	HART Unit
JOULE	newton meter	1126	69
DEKATHERM		1184	89
ft-lb	foot pound force	1185	126
kWh	kilowatt hour	1179	128
Mcal	mega calorie	1182	162
MJ	mega joule	1172	164
Btu	british thermal unit	1183	165

7.15.12 Power

Keyword	Description	FF / PA Unit	HART Unit
kW	kilo watt	1190	127
Hp	horsepower	1198	129
Mcal/h	mega calorie per hour	1195	140
MJ/h	mega joule per hour	1196	141
Btu/h	british thermal unit per hour	1197	142

7.15.13 Force

Keyword	Description	FF / PA Unit	HART Unit
NEWTON		1120	68

7.15.14 Electrical

The table contains units for various electrical .

Keyword	Description	FF / PA Unit	HART Unit
mV	millivolt	1243	36
mA	milliampere	1211	39
Ohm		1281	37
uS	microsiemens	1290	56
VOLT		1240	58
pF	picofarad	1249	153
kOhm	kiloOhm	1284	163

7.15.15 Miscellaneous

Keyword	Description	FF Unit	PA Unit	HART Unit
HERTZ		1077		38
cSt	centistokes, kinematic viscosity	1164		54
cP	centipoise, absolute viscosity	1162		55
pH	acidity or alkalinity of solution	1422		59
mS/cm	milli siemens per centimeter	1302		66
uS/cm	micro siemens per centimeter	1586	1552	67
degBrix	degrees brix	1426		101
sol/wt	percent solids per weight	1343		105
sol/vol	percent solids per volume	1344		106
degBall	degrees balling	1427		107
proof/vol	proof per volume	1428		108
proof/mass	proof per mass	1429		109
PPM	parts per million	1423		139
DEGREE		1005		143
RADIAN		1004		144
VolPercent	volume percent	1669	1562	149
PercentStmQual	percent steam quality	1345		150
mL/L	milliliters per liter	1673	N.A.	154
uL/L	microliters per liter	1678	N.A.	155
dB	decibel	1383		156
PLATO	percent plato	1346		160
LEL	percent lower explosion level	1670	N.A.	161
PPB	parts per billion	1424		169

7.16 PARENT_CLASS and CLASS Keywords

7.16.1 Physical block PARENT_CLASS Keyword list

PB PARENT_CLASS Keyword		
TRANSMITTER	or	1
ACTUATOR	or	2
DISCRETE_IO	or	3
CONTROLLER	or	4
ANALYZER	or	5
LAB_DEVICE	or	6
MULTI-VARIABLE	or	127
UNUSED	or	250

7.16.2 Transducer block PARENT_CLASS and CLASS Keyword list

TB PARENT_CLASS Keyword		CLASS Keyword	
PRESSURE	or 1	DIFFERENTIAL	or 1
		ABSOLUTE	or 2
		GAGE or GAUGE	or 3
		PRESSURE_LEVEL_FLOW	or 4
		PRESSURE_LEVEL	or 5
		PRESSURE_FLOW	or 6
		MIXED_ABSOLUTE_DIFF_PRESSURE	or 7
		UNUSED	or 250

TB PARENT_CLASS Keyword		CLASS Keyword	
TEMPERATURE	or 2	THERMOCOUPLE	or 1
		RESISTANCE_THERMOMETER	or 2
		PYROMETER	or 3
		THERMOCOUPLE_DC_VOLTAGE	or 16

TB PARENT_CLASS Keyword	CLASS Keyword	
	THERMOMETER_RESISTANCE	or 17
	THERMOCOUPLE_THERMOMETER_RESISTANCE_DC_VOLTAGE	or 18
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
FLOW or 3	ELECTROMAGNETIC	or 1
	VORTEX	or 2
	CORIOLIS	or 3
	THERMAL_MASS	or 4
	ULTRASONIC	or 5
	VARIABLE_AREA	or 6
	DIFFERENTIAL_PRESSURE	or 7
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
LEVEL or 4	HYDROSTATIC	or 1
	ECHO_LEVEL	or 2
	RADIOMETRIC	or 3
	CAPACITY	or 4
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
ACTUATOR or 5	ELECTRIC	or 1
	ELECTRO-PNEUMATIC	or 2
	ELECTRO-HYDRAULIC	or 3
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
DISCRETE_IO or 6	SENSOR_INPUT	or 1
	ACTUATOR	or 2
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
ANALYZER or 7	STANDARD	or 1
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
AUXILIARY or 8	TRANSFER	or 1
	CONTROL	or 2
	LIMIT	or 3
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
ALERT or ALARM or 9	BINARY_MESSAGE	or 1
	UNUSED	or 250

TB PARENT_CLASS Keyword	CLASS Keyword	
UNUSED or 250	UNUSED	or 250

Softing Industrial Automation GmbH

Richard-Reitzner-Allee 6
85540 Haar / Germany
https://industrial.softing.com

+ 49 89 45 656-340
info.automation@softing.com