FIELDBUS FOUNDATION Field Device Software
PROFIBUS PA Device Software

Capabilities and Interfaces

Version 2.41

Date: November, 2011

SOFTING Industrial Automation GmbH
Richard-Reitzner-Allee 6
D-85540 Haar
Phone (++49) 89 45656-0
Fax (++49) 89 45656-399

Table of Contents

List of figures

List of tables

1 References

MAKE-FDSW	Generation Process User Manual Version 2.41 (MAKE_Desc.pdf)
PA-302	PROFIBUS Profile for Process Control Devices, Version 3.02

2 Terms and Abbreviations

APPL	Application Layer
CTK	Conformance Test Kit
FBLK	Function Block
FBL	Function Block Layer
FBS	Function Block Shell
FD	Field Device
GenVFD	Generate VFD tool by Softing
ID	Identifier
IF	Interface
ITK	Interoperable Test Kit
NV	Non-volatile memory
OS	Operating System
OSIF	Operating System Interface
VFD	Virtual Field Device
PDU	Process Data Unit
MVC	Abbreviation for Multi Variable Container (FF only)

3 Introduction

This document describes capabilities and interfaces of Softing's Fieldbus Foundation / PROFIBUS PA field device software, version 2.41.

There are chapters describing the differences between version 2.30 and the previous versions 2.2x and 2.12.

Version 2.41 was tested with a CPU frequency of 4MHz and Renesas M16C 30624N / 30626P CPU.

4 Capabilities of the Field Device Software

The following tables describe capabilities, the supported function blocks and the quantity structure. The quantity structure is fixed to an amount of resources, which will fit most of the applications. For larger applications (high amount of function blocks and thereby need of more resources) the quantity structure can be changed.

4.1 FF Capabilities

This chapter describes the capabilities and quantity structure of the FF field device software.

4.1.1 FF Features

The next table describes the features of the FF field device software. All supported features can be set by compilation. Please refer to [MAKE-FDSW] and the configuration file fdev_cfg.h.

Capabilities	Selection	Supported	Meaning
Stack type	Link Master / Basic Device	S (*)	The device can act as Link Master or Basic Device.
Hard Types	Scalar Input	S	The acts as sensor and delivers an analogue input value with float and status.
	Scalar Output	S	The device acts as actuator and uses an analogue output value with float and status.
	Discrete Input	S	The acts as sensor and delivers a discrete input value with integer and status.
	Discrete Output	S	The device acts as actuator and uses a discrete output value with integer and status.
Resource features	Unicode Strings	N	-
	Reports	S	The report of alarms and events is supported
	Fault State	S	Fault state behavior of out function blocks.
	Soft Write Lock	S	If Soft Write Lock is enabled write access to configuration parameters will be rejected. Soft Write Lock will be enabled if in FEATURE_SEL the appropriate feature bit is set and if the WRITE_LOCK parameter is set to LOCKED.
	Hard Write Lock	S	If Hard Write Lock is enabled write access to configuration parameters will be rejected. Hard Write Lock will be enabled if the appropriate feature bit in FEATURE_SEL is set and if the write lock jumper is set.

Capabilities	Selection	Supported	Meaning
	OUT Readback	S	Read back feature of DO and AO function blocks.
	Direct Write to Output parameter	N	-
	Report MVC	S (**)	This MVC is used for reports. Can be used for contained, input and output parameters.
	Change of BYPASS in an automatic mode	N	-
	Publisher/Subscriber MVC	S (**)	This MVC s used to publish and subscribe input and output parameters.
	Multi Bit Alarm	S	Each bit-alarm of a discrete alarm parameter is reported individually
	Restart/Relink required after using FB_Action	N	-
	Deferral of Inter-Parameter Write Checks	N (***)	-

Table 1 FF stack capabilities

S – means supported
N – means not supported

(*) Link master functions are optional. Link master functions require additional software modules which are not part of the standard FF field device software.

(**) Support of MVC objects is an optional feature. MVCs require additional software modules which are not part of the standard FF field device software.

(***)For this feature FF-891 states: "The following is true if the “Deferral of Inter-Paramter Write Checks” bit is true in the features bit string. Whenever a function block MODE_BLK target is Out-of-Service or transducer block MODE_BLK target mode is Out-of-service, the block must not perform or report inter-parameter error checks. If the resource block target mode is Out-of-service, then the resource block and function blocks in the device must not perform or report inter-parameter error checks. On transition to an in-service target mode (Man, Auto, Cas, RCas,or ROut), the inter-parameter error checks must be performed."
This behavior described in FF-891 should be expected to be 'normal' for all blocks. It seems not to be useful to allow disabling "Deferral of Inter-Parameter Write Checks". Therefore this feature is 'not supported' which actually means it is always enabled and can not be disabled.

4.1.2 FF Quantity Structure

This chapter describes the standard quantity structure of the FF field device software. It will fit for most FF applications. For complex application, it is possible to enlarge the quantity structure for some of the object groups. Please contact Softing if your application requires a larger quantity structure.

Part	Default setting	Meaning
Max VCR entries	1 for management VFD (fixed)	This VCR is fixed and used by any host system to configure the device.
	23 for FBAP VFD	VCR which can be configured by a host system.
Max link objects	22	The maximum number of link objects in the device.
Max tag length	32 / 16	The maximum length of tags. By default the tag length is 32 bytes. The length can be changed to 16 bytes by setting the flag tag_len_16. Refer to function appl_init().
Max PDU length	256	Maximum number of bytes in a PDU.
Max number of blocks	32	The maximum number of blocks (resource block + function blocks + transducer blocks).
Trend objects	40 (may be limited by heap and non-volatile memory restrictions)	
Max MVCs (*)	16 (may be limited by heap and non-volatile memory restrictions)	The maximum number of Multi Variable Containers.
Max number of elements per MVC (*)	32	

Table 2 Quantity structure FF field device software

(*) Support of MVC objects is an optional feature. MVCs require additional software modules which are not part of the standard FF field device software.

4.1.3 Function blocks supported

FF function blocks supported.

Function block type	Profile revision	Execution time [ms] *
Analogue Input (AI)	0x0101	30
Analogue Output (AO)	0x0102	30
Discrete Input (DI)	0x0103	20
Discrete Output (DO)	0x0104	30
Proportional-Integral-Derivate (PID)	0x0108	40
Multiple Analog Input (MAI)	0x0130	50
Multiple Analog Output (MAO)	0x0131	50
Multiple Discrete Input (MDI)	0x012E	50
Multiple Discrete Output (MDO)	0x012F	50
Arithmetic (AR)	0x0127	60
Input Selector (IS)	0x0126	30
Integrator (IT)	0x0120	60
Signal Characterizer (SC)	0x011D	40

Table 3 Supported FF function blocks

Note: Execution times were measured with a Renesas M16C/62P CPU at 4MHz clock frequency.

4.2 PA Capabilities

This chapter describes features and quantity structure of the PA device software.

4.2.1 PA Features

The next table describes the different capabilities of the PA device software. All features can be set by compilation. Please refer to [MAKE-FDSW] and the configuration files pdev_cfg.h and dps_cfg.h.

4.2.2 Function blocks supported

PA function blocks supported.

Function block type	Parent class	Class	Execution time [ms] *
Analog Input (AI)	1	1	50
Analog Output (AO)	1	1	50
Discrete Input (DI)	2	2	50
Discrete Output (DO)	2	2	50
Totalizer (TOT)	5	8	50

Table 4 Supported PA function blocks

* Execution times were measured with a Renesas M16C/62P CPU at 4MHz clock frequency.

4.2.3 Quantity Structure of PA Device Software

Part	Default setting	Meaning
MSAC C1 services	Not defined.	C1 services are not supported.
MSAC C2 services	Defined.	C2 services are supported.
Max. number of parallel C2 services	3	Maximal number of possible parallel C2 services.
Max. size of user data in the PDU.	128	By default for cyclic and acyclic services.
Max. possible I/O data size	Inputs: 48 bytes for SPC42 244 bytes for UFC100 and Find1+ Outputs: 32 bytes for SPC42 244 bytes for UFC100 and Find1+	The maximal size for inputs and outputs depends on the fieldbus controller chip.

Table 5 Quantity structure PA device software

4.3 Configuration File fdev_cfg.h

This file contains configuration settings of the FF field device.

`#define DEV_SERIAL_NR`
Each device requires a unique device-id. The device-id is a visible string. Its length is 32 octets. In Softing's field device software the device-id has three sections: 6 octets for the manufacturer-id, 4 octets for the device type and 22 octets for a unique serial number.

In the module appl_if.c *DEV_SERIAL_NR* is stored in a special 32+1 byte large constant data segment at begin of the flash area (segment FW_INFO). There are 32 bytes of serial number + '\0' termination. In the .hex file this segment may be overwritten with an external tool to get from an initial .hex file a set of .hex files, each with a unique serial number.

In Softing's ROCK application the last 22 octets of *DEV_SERIAL_NR* are used as a dummy serial number.

By means of the example routine *appl_if.c::appl_get_serial_number()* it is shown how the serial number is propagated to the communication stack. The communication stack uses the serial number to build the device-id and a default PD-tag. The FB shell uses the last 16 octets of the serial number to build default block tags.

In a real field device the serial number is typically provided by a device-specific application object.

Please note that Softing's FBK and FBK2 platforms have got a PROM holding a unique serial number. The communication stack reads this unique serial number and overwrites the application's serial number by the serial number stored in the PROM.

```
#define NV_RAM_CYCLE_TIME            0UL    /* Unit [s]. */
#define EXTRA_NV_RAM_CYCLE_TIME      0UL    /* Unit [s]. */
```
These macros determine the time in seconds for the cyclic update of the non-volatile and extra non-volatile memory. Set to 0 means, that not cyclic update will be performed.

`#define DEV_CONFIG_REVISION`
The device config revision is used in Softing's example application to optionally reset the whole NV-RAM. When the device starts the first time it stores `DEV_CONFIG_REVISION` in NV-RAM. During each subsequent start the application checks `DEV_CONFIG_REVISION` from fdev_cfg.h against the stored device config revision. If there is a mismatch the whole NV-RAM is reset and the device starts with default configuration data.

`#define RES_VAL_xxx`
These macros determine the initial or constant values of resource block parameters. They are applied in GenVFD scripts.

Please note that the macros
```
/*#define RES_VAL_DEV_TYPE          … -> defined in makeconfig.mak   */
/*#define RES_VAL_DEV_REV           … -> defined in makeconfig.mak   */
```
are defined in makeconfig.mak

```
#define DEV_DEFAULT_PD_TAG              "ROCK APPLICATION"  /* max. 16 bytes */
```

When the device starts with default configuration data this macro is used to build a default PD tag. The communication stack builds the default PD tag from the 16 octets of DEV_DEFAULT_PD_TAG and the last 16 octets of the device serial number.

#define DEV_SM_OPERATIONAL
This macro is used to set the 'sm_operational' flag in the T_FDC_DEVICE_START structure. The flag is relevant if the device starts with default configuration data. Setting this macro to TRUE means the device starts with a PD tag and a node address though no host set a PD tag or a node address. 'sm_operational = TRUE' could be useful in device development phase; when devices are delivered to customers 'sm_operational' should be set to FALSE.

#define DEV_DEFAULT_NODE_ADDRESS
This macro is used to determine the default node address in case 'sm_operational' is set to TRUE.

#define DEV_DEFAULT_DEV_TYPE
This macro is used to set the 'default_dev_type' flag in the T_FDC_DEVICE_START structure. The flag relevant for link master devices if they start with default configuration data. The flag determines whether a link master device should function as link master or basic device.

For basic devices the flag is not relevant.

#define DEV_ST_MIN 8
Setting for the slot time.

#define DEV_MID_MIN 6
Setting for the min. inter PDU delay.

#define DEV_MRD_MIN 5
Setting for the max. response delay.

The macros DEV_ST_MIN, DEV_MID_MIN and DEV_MRD_MIN determine the 'fastest' busparameters the device is able to work with. The values given in fdev_cfg.h were tested with Softing's HW-platforms running with Softing's example application. For different HW-platforms and real application layers it has to be tested if the device is able to work with these busparameters.

4.4 Configuration Files for PA

The following files are special for PA.
…\target\cfg\dps_cfg.h
…\target\cfg\pdev_cfg.h

4.4.1 File dps_cfg.h

This file describes specific settings for the DP Slave.

#define MSAC_C2_SUPPORTED
With this setting the class 2 services are supported by the DP slave. Do not change it for PA.

#define DPS_MAX_DATA_LEN 128
That is the maximal data length for one telegram. Note, that the maximal length can be 244 bytes!

```
//#define DPS_WD_TIMEOUT_EVENT_SUPPORTED     Currently not supported.
//#define DPS_NEW_DATA_EVENT_SUPPORTED       Currently not supported.
//#define DPS_STATE_CHANGE_EVENT_SUPPORTED   Currently not supported.
```
Do not define any of these macros! They are currently not supported!

#define DPS_MAX_NUMBER_ACC2 3

Defines the maximum number of parallel usable acyclic class 2 services. Do not change, if there is no need.

`#define DP_TIMER_SUPPORTED`
Do not change.

4.4.2 File pdev_cfg.h

This file is the analog to the FF file fdev_cfg.h. The file contains the configuration information for a PROFIBUS PA device.

`#define DEV_CONFIG_REVISION`
The device config revision is used in Softing's example application to optionally reset the whole NV-RAM. When the device starts the first time it stores `DEV_CONFIG_REVISION` in NV-RAM. During each subsequent start the application checks `DEV_CONFIG_REVISION` from pdev_cfg.h against the stored device config revision. If there is a mismatch the whole NV-RAM is reset and the device starts with default configuration data.

```
#define DEV_PA_PROFILE_REVISION      0x302 /* PA profile revision 3.02.
                                            Copied into each block header */
```
This macro declares the supported PROFIBUS PA profile revision. Softing's PROFIBUS PA device software. Version 2.41 supports the PA Profile, V3.02.

`#define PHY_VAL_xxx`
These macros determine the initial or constant values of physical block parameters. They are applied in GenVFD scripts.

```
#define PHY_VAL_IDENT_NO_SEL         127
```
This macro defines the initial ident number selector of the device. Please, refer to the PROFIBUS PA specification [PA-302] for more information on the ident number selector. [PA-302] requires ident number selector 127 (adaption mode) as initial value.

`#define PHY_VAL_FEATURE`
This macro defines the supported and the initially enabled features of the device. The PA device V2.41 supports the features "Condensed status and diagnosis" and "Classic status and diagnosis".

```
#define DPS_MIN_TSDR                 11         /* min TSDR                 */
```
This macro describes the min TSDR supported by the DP slave

```
#define NV_RAM_CYCLE_TIME            0UL    /* Unit [s]. */
#define EXTRA_NV_RAM_CYCLE_TIME      0UL    /* Unit [s]. */
```
These macros determine the time in seconds for the cyclic update of the non-volatile and extra non-volatile memory. Set to 0 means, that not cyclic update will be performed.

`#define DPS_VENDOR`
The vendor name is a visible string with a maximum length of 32 bytes. It is used to build the FDL identification string of the DP-slave.

`#define DPS_CONTROLLER_TYPE`
Controller type is a visible string with a maximum length of 32 bytes. It is used to build the FDL identification string of the DP-slave.

`#define DEV_SERIAL_NR`
The physical block parameter DEVICE_SER_NUM requires a unique serial number for each device.

In the module appl_if.c the serial number is stored in a special 16+1 byte large constant data segment at begin of the flash area (segment FW_INFO). There are 16 bytes of serial number + '\0' termination. In the .hex file this segment may be overwritten with an external tool to get from an initial .hex file a set of .hex files, each with a unique serial number.

In Softing's ROCK application the macro DEV_SERIAL_NR is used as a dummy serial number.

By means of the example routine *appl_if.c::appl_get_serial_number()* it is shown how the serial number is propagated to the communication stack.

In a real field device the serial number is typically provided by a device-specific application object.

Please note that Softing's FBK and FBK2 platforms have got a PROM holding a unique serial number. The communication stack read this unique serial number and overwrites the application's serial number by the serial number stored in the PROM.

`#define ID_NO_FOR_SEL_x`
These macros are used to determine the ident numbers associated with different ident number selectors.

Please note that with the new automatic ident number adaptation there is typically more than one ident number associated with the ident number selector 127 (= adaption mode)

`#define PA_DEV_REV`
`#define PA_DEV_REV_COMP`
These macros are used to describe the compatibility with other versions of the device. The revision numbers are copied into each block header. The meaning of these revision numbers is described in [PA-302], chapter 5.5.4.2.1 'Device parameters' and chapter 5.5.4.3 'Compatibility rules'.

5 Application Layer

The FF/PA field device software consists of a communication stack, a function block shell, a function block layer and an application layer.

The application layer controls the device startup phase and implements the interface to sensors and actuators.

The components target\fdev (FF) and target\pdev (PA) implement examples of application layers.

5.1 FF Application Layer

The *fdev* component provides the following modules:

appl_if.c
This module contains functions that control the device startup phase

appl_ifd.c
The software download class 3 is performed by a special field device software with reduced functionality. This reduced field device software is designated as download device software. appl_ifd.c implements functions which controls the startup phase of the download device software. The functions work similar to the standard startup routines in module appl_if.c.

appl_res.c
This module contains examples of routines that implement the application-related part of the resource block.

appl_trn.c
This module contains interface functions of transducer blocks. Each transducer block has to provide a generic set of interface functions which are called by the function block shell and the function block layer.

appl_dia.c
With FF-912, Field Diagnostics Profile, a new application-related functionality is available. appl_dia.c shows in a simple example how the application can use field diagnostics.

appl_dlh.c
This module contains functions for software download. It implements an example of a domain download handler. The download handler is the counter part of the download server which is a part of the System Management Agent.

appl_nrv.c
This module contains functions for a background scan of the NV-RAM.

5.1.1 Initialization and Configuration of the FF application

The functions to initialize the FF application and to configure the communication stack are part of module appl_if.c. The systems startup phase can be divided into two parts. First the initialization sequence which is executed not in task context and the configuration sequence, which is executed in task context. When the OS scheduler starts its operation and the tasks are scheduled according their priority, each task will get a system start event first. This event is used to initialize the FF application and to configure the FF communication stack before the normal operation starts. Refer to the following picture:

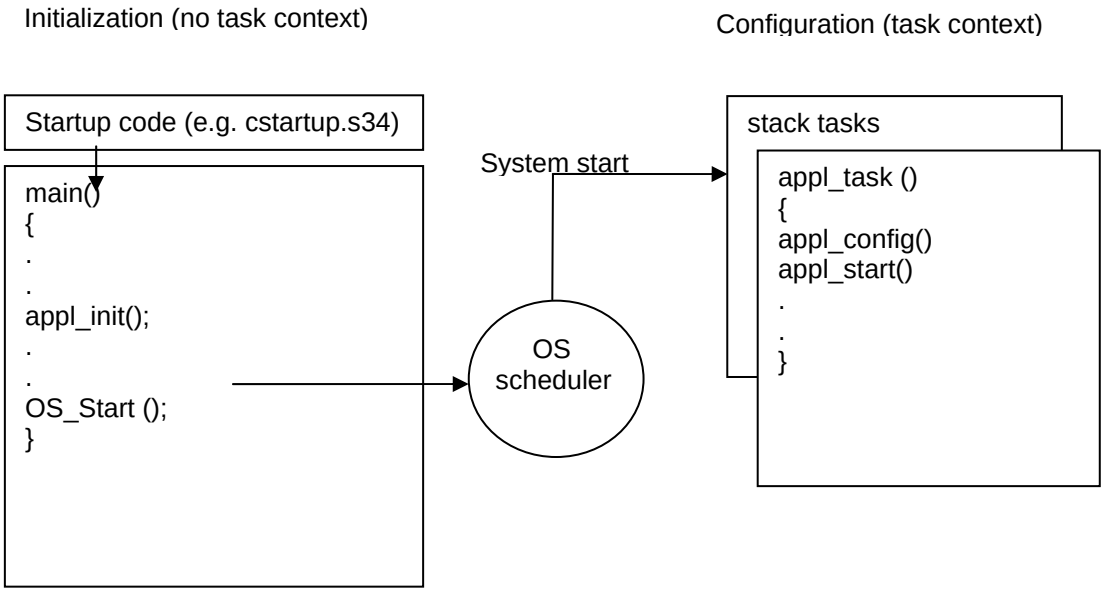

Figure 1 Initialization and Configuration of the FF application

5.2 PA Application Layer

The *pdev* component provides the following modules:

appl_if.c
This module contains functions that control the device startup phase

appl_phy.c
This module contains examples of routines that implement the application-related part of the physical block.

appl_trn.c
This module contains interface functions of transducer blocks. Each transducer block has to provide a generic set of interface functions which are called by the function block shell and the function block layer.

appl_dia.c
With [PA-302] condensed status becomes mandatory. appl_dia.c shows in a simple example how the application can implement condensed status handling.

appl_nrv.c
This module contains functions for a background scan of the NV-RAM.

5.2.1 Initialization and Configuration of the PA application

The functions to initialize the PA application and to configure the DP slave are in module appl_if.c. The systems startup phase can be divided into two parts. First the initialization sequence which is executed not in task context and the configuration sequence, which is executed in task context. When the OS scheduler starts its operation and the tasks are scheduled according their priority, each task will get an system start event first. This event is used to initialize the PA application and to configure the DP slave before the normal operation starts. Refer to the following picture:

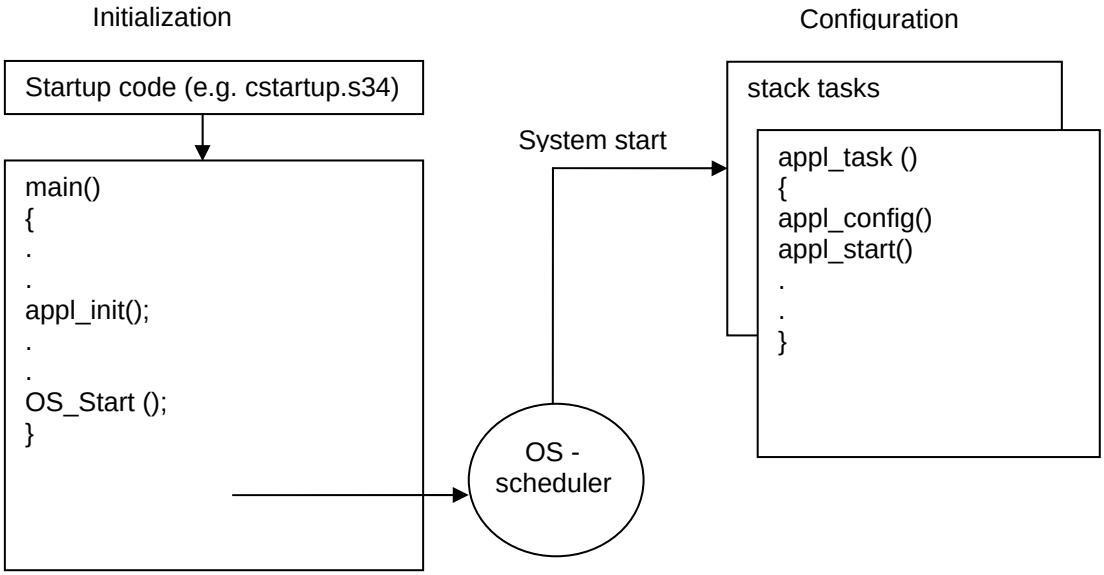

Figure 2 Initialization and Configuration of the PA application

6 Differences Between Field Device, Version 2.12 and Version 2.30

The Softing FF/PA field device software, version 2.30 is a further stage of version 2.12 and 2.2x. Version 2.30 contains some additional features and bug fixes.

This chapter describe new features of version 2.30 and how to move easily from version 2.12 and 2.2x to version 2.30.

IAR C - Compiler for the M16C version 3.50 is the new standard compiler and is set as default in the Makeconfig.mak files.

6.1 Differences in FF

6.1.1 New features in version 2.30 compared to version 2.12 (FF)

The FF field device software, version 2.30 has the following new features.

New features	Description
Multi Bit Alarm	Refer to FF-890-1.6 for more information.
Multi Variable Container for Reports	Refer to FF-890-1.6 for more information.
Multi Variable Container for Publishing and Subscribing.	Refer to FF-890-1.6 for more information.
Non-volatile block parameters (N-parameters are stored in a separate block in the non-volatile memory.	In version 2.12 N-parameters were stored in the same area in the non-volatile memory (EEPROM, FRAM, …) as the parameters marked with the static revision – flag (S-flag) for the specific block (resource, function or transducer). In version 2.30 all N-parameters are stored in one area in the non-volatile memory.
Extra non-volatile data (currently used by the integrator function block).	The extra non-volatile data are important information of blocks, which should not be lost during restart of the device. Currently this feature is used for the integrator function block. The storage of these data in the extra-non volatile memory is done cyclically, when defined in file fdev_cfg.h.
Storing the extra non-volatile memory on power fail.	This feature can be used during power fail, when the processor supports the power fail detection. Please refer to file hw_intr.c for more information. Note, that this depends on the device hardware (processor, kind of non-volatile memory,…) and must be customized.
Double buffering of N-parameters and extra non-volatile data.	N-Flag parameters as well as extra-non-volatile data are stored in two separate areas. This feature prevents data loss in case of power fail during non-volatile memory write access.

Table 6 New features in version 2.30 compared to version 2.12 (FF)

6.1.2 How to migrate from version 2.12 to version 2.30 (FF)

This chapter describes the step to migrate from version 2.12 to version 2.30 easily.

6.1.2.1 Generation environment (FF)

In file MakeConfig.mak are two more entries. The settings of these entries should be checked to make sure that it fits to the application and the device. The entries are:

MVC_TYPE

MVC are supported in version 2.30. Set it to USE_MVC to support this feature by the device. Please note that the support of MVC objects is an additional feature. MVC objects are not supported by the standard field device software.

NV_DATA_STORAGE

Double buffering of non-volatile memory and extra-non-volatile memory is supported in version 2.30. Set it to NV_DATA_DOUBLE_BUFFER to enable this feature. Do not enable this feature (set to NV_DATA_SINGLE_BUFFER), when the non-volatile memory driver supports this behavior by itself!

6.1.2.2 Modifications in the device startup phase (FF)

Some structures and functions, which are used by the application interface were changed, deleted or added. The verification of the non-volatile memory has changed and can be enabled or disabled by compilation. There is a new way how different parts of the field device software apply for non-volatile memory. A new feature in version 2.30 is the usage of so called extra-non-volatile memory and the calculation of the size of extra-non-volatile memory. The initialization is part of file appl_if.c.

Function appl_init()

The global variable ‘pass_itk’ is used to change the behavior to fulfill the requirements of ITK 5.0.1. Please see the comment in function appl_init().

Function appl_task()

The appl task does not use a timer in version 2.30. The cyclic verification of the NV RAM is done by the background task now. The timer event is not used any more.

Function appl_config()

The applyment and check of the NV RAM is separated into different components. New is the calculation of the extra NV RAM, which is currently used by the integrator function block.

Function appl_start()

The function takes new roles during the start of the device. Please refer to the code of the function.

Function appl_check_nv_ram()

This function replaces the function appl_check_config().

Function appl_apply_for_nv_ram()

This function is new and replaces function appl_load_new_config(). Functionality previously in function appl_load_new_config() moved although to function appl_load_new_fdc_config(). Please refer to the mentioned function in version 2.12 and version 2.30.

Function appl_calc_extra_nv_ram_size()

This function is new and used to calculate the extra NV RAM size. Currently used by the integrator function block. Please refer to the code of the function.

Function appl_store_appl_ident()

This function replaces function appl_load_device_ident(). Please refer to the code of new function.

Function appl_load_new_fdc_config()

New function. Parts are adapted from the old function appl_load_new_config(). Please refer to code the function.

Function appl_get_serial_number()
The calculation of the loop_count variable offers two choises. Disable the unused one.

Function appl_synchronise_fbap_with_device()
This function is new. Please refer to the code of the function.

Function appl_provide_device_ident_to_fdc()
This function is new in version 2.30. The function is used to load specific resource block parameters (Manufacturer ID, Device Type and Min. Cycle Time) to the FF communication stack.

Define ENABLE_NV_RAM_VERIFY
If defined the cyclic NV RAM verification is enabled. The define NV_RAM_VERIFY_CYCLE_TIME is used for the specification of the cycle time (unit 100ms).

Data structure T_FBS_CALL_FDEV_FUNCT
Most parts come from structure T_FBIF_FUNCT, which was used in version 2.12. The component a_of_chan_unit_check moved into the new structure T_FFBL_CALL_FDEV_FUNCT. The structure component a_of_verify_nv_ram is used to set the call back function when the cyclic verification of the non-volatile memory is enabled.

Data structure T_FFBL_CALL_FDEV_FUNCT
This data structure is new. Currently the structure has one component, a_of_chan_unit_check, which is used to store the pointer of the call back function to check the write of the channel parameter for the analogue input and analogue output function block.

Cyclic verification of the non-volatile memory
The cyclic verification of the non-volatile memory can be enabled by “#define ENABLE_NV_RAM_VERIFY” in file appl_if.c. If the cyclic verification is enabled, the background task, which is executed every 100ms, calls the callback function for verification. Please refer to function appl_init() and appl_verifiy() for more information.

Calculation of the extra-non-volatile memory
This is new in version 2.30 and currently used only for the integrator function block. Please refer to function appl_calc_extra_nv_ram_size().

6.1.2.3 Transducer blocks and their methods (FF)

The functions or methods of the transducer blocks are located in file appl_trn.c. The differences between version 2.12 and version 2.30 are:

Function Appl_trn_init()
This function is deleted in version 2.30.

Function Appl_check_channel_unit()
Used only when an AO and AI function block is part of the device. The code of the function is enabled when AI_BLOCK_AVAILABLE or AO_BLOCK_AVAILABLE is defined.

Function Appl_check_block_state()
The function is used only, when the device uses block instantiation. The code of the function is enabled, when FBL_BLOCK_INST is defined.

Function Appl_trn_update_modes()

This function is deleted in version 2.30. The update of the transducer block mode is done by the back ground method of the transducer block.

Function Appl_trn_pending_response_demo()
This function was deleted in version 2.30.

Function Appl_trn_handle_pending_response()
This function was deleted in version 2.30.

Function Appl_update_trn_mode()
The prototype of the function has changed.

Function appl_trn_start_tb()
This function is called from the specific transducer block start method. Common code of the specific transducer block start methods moved into this function.

Function Background_XXXXTB()
This callback function is new in version 2.2x. The back ground function of the specific transducer block is called by the back ground task periodically (every 100ms). It can be used to perform block-related background actions.

Structure T_FBIF_WRITE_DATA
The structure has got two new components. The component “startup_sync” is TRUE if a write access is used to synchronize an object of the application with an transducer or resource block parameter. In this case the new value is copied to the block parameter but the static revision counter is not incremented and no update event is generated. Usually the flag is TRUE when a parameter with a GenVFD V-flag is written in the startup phase. For a 'normal' write accesses the flag is always FALSE. The second new component is named “phase”. The phase of the write handling can be PRE_WRITE_CHECK or POST_WRITE_ACTION.

6.1.2.4 Resource block and its methods (FF)

The functions and methods for the resource block are in appl_res.c. The differences between 2.12 and 2.30 are:

Function Appl_background_RESB()
The function Appl_execute_RESB in version 2.12 was replaced with function Appl_background_RESB().

Function Appl_write_handler_RESB()
In version 2.30 this function interprets the new component “phase” of the structure T_FBIF_WRITE_DATA. It distinguishes between pre and post write action. Please refer to the description of the structure T_FBIF_WRITE_DATA in the previous chapter and to the function in module appl_res.c

6.2 Differences in PA

6.2.1 New features in version 2.30 compared to version 2.12 (PA)

Version 2.30 has against version 2.12 the following new features.

New features	Description
Non-volatile block parameters (N-parameters) are stored in a separate block in the non-volatile memory.	In version 2.12 N-parameters were stored in the same area in the non-volatile memory (EEPROM, FRAM, …) as the parameters marked with the static revision – flag (S-flag) for the specific block (physical, function or transducer). In version 2.30 all N- parameters of the blocks are stored in one area in the non-volatile memory.
Extra non-volatile data (currently used by the integrator function block).	The extra non-volatile data are used to store important information of blocks, which should not be lost during restart of the device. Currently this feature is used for the integrator function block. The store of these data in the extra-non volatile memory is done cyclically, when defined in file pdev_cfg.h.
Storing the extra non-volatile memory on power fail.	This feature can be used during power fail, when the processor supports the power fail detection. Please refer to file hw_intr.c for more information. Note, that this depends on the device hardware (processor, kind of non-volatile memory,…) and must be customized.
Double buffering of N-parameter and extra non-volatile data.	N-Flag parameters as well as extra-non-volatile data are stored in two separate areas. This feature prevents data loss in case of power fail during non-volatile memory write access.
Condensed Status	Condensed status and diagnosis information according to the Amendment 2 to PROFIBUS profile for process control devices V3.01.

Table 7 New features in version 2.30 compared to version 2.12 (PA)

6.2.2 How to migrate from version 2.12 to version 2.30 (PA)

This chapter describes the step to migrate from version 2.12 to version 2.30 easily.

6.2.2.1 Generation environment (PA)

In file MakeConfig.mak is one more entry. The settings of this entry should be checked to make sure that it fits to the application and the device. The entry is:

NV_DATA_STORAGE

Double buffering of non-volatile memory and extra-non-volatile memory is supported in version 2.30. Set is to NV_DATA_DOUBLE_BUFFER to enable this feature. Do not enable this feature (set to NV_DATA_SINGLE_BUFFER), when the non-volatile memory driver supports this behavior by itself!

6.2.2.2 Modifications in the device startup phase

Some structures which are used by the application interface have changed or were added. The verification of the non-volatile memory has changed and can be enabled or disabled by compilation. There is a new way how different parts of the field device software apply for non-volatile memory. A new feature in version 2.30 is the usage of so called extra-non-volatile memory and the calculation of the size of extra-non-volatile memory. The initialization is part of file appl_if.c.

Structure T_FBIF_FUNCT
The structure is not used any more and the components are replaced by structure T_FBS_CALL_PDEV_FUNCT and T_PFBL_CALL_FUNCT.

Structure T_FBS_CALL_PDEV_FUNCT
New structure developed from structure T_FBIF_FUNCT. The component a_of_approve_id_no is not part of the structure any more. The component a_of_verify_nv_ram is new. The call back functions are called by the function block shell.

Structure T_PFBL_CALL_PDEV_FUNCT
This structure is new. The call back functions are called by the function block layer.

Cyclic verification of the non-volatile memory
The cyclic verification of the non-volatile memory can be enabled by “#define ENABLE_NV_RAM_VERIFY” in module appl_if.c. If the cyclic verification is enabled, the background task, which is executed every 100ms, calls the callback function for verification. Please refer to function appl_init() and appl_verifiy() for more information.

Calculation of the extra-non-volatile memory
This is new in version 2.30 and currently used only for the totalizer function block. Please refer to function appl_calc_extra_nv_ram_size().

Module appl_diag.c
This module is new. Its used for condensed status and diagnostic calculation.

6.2.2.3 Transducer blocks and their methods

The functions or methods of the transducer blocks are in file appl_trn.c. The differences between version 2.12 and version 2.30 are:

Function Appl_check_channel_unit()
This function is new. The function is used only, when an AI or AI function block is part of the device. The code of the function is enabled when AI_BLOCK_AVAILABLE or AO_BLOCK_AVAILABLE is defined.

Function Background_XXXXTB()
This callback function is new in version 2.30. The back ground function of the specific transducer block is called by the back ground task periodically (every 100ms). It can be used to perform block-related background actions.

Structure T_PBIF_WRITE_DATA

The structure has got two new components. The component “startup_sync” is TRUE if a write access is used to synchronize an object of the application with an transducer or physical block parameter. In this case the new value is copied to the block parameter but the static revision counter is not incremented and no update event is generated. Usually the flag is TRUE when a parameter with a GenVFD V-flag is written in the startup phase. For a 'normal' write accesses the flag is always FALSE. The second new component is named “phase”. The phase of the write handling can be PRE_WRITE_CHECK or POST_WRITE_ACTION.

6.2.2.4 Physical block and its methods

The functions and methods for the resource block are in appl_phy.c. The differences between 2.12 and 2.30 are:

Function Appl_write_handler_PHYB
The function distinguishes between the two phases PRE_WRITE_CHECK and POST_WRITE_ACTION.

Function Appl_execute_handler_PHYB
Modifications in the scope of setting the slave address. Refer to the description in the function header.

7 Differences between version 2.2x and version 2.30

The FF/PA field device software, version 2.30 is an upgrade of the FF/PA field device software, version 2.21. Version 2.30 contains some additional features and bug fixes.

This chapter describes the new features of version 2.30 and how to move from version 2.21 to version 2.30.

7.1 Differences in FF

7.1.1 New features in version 2.30 compared to version 2.2x (FF)

The FF field device software version 2.30 has the following new features.

New features	Description
Field Diagnostics	Support of Field Diagnostics in accordance with FF-912.

Table 8 New features in version 2.30 (FF)

7.1.2 How to migrate from 2.2x to 2.30 (FF)

This chapter describes the step to migrate from version 2.2x to version 2.30 easily.

7.1.2.1 Generation environment (FF)

The file Makeconfig.mak got the entry SW_DOWNLOAD_TYPE. The setting of this entry should be checked to make sure that it fits to the application and the device. Furthermore the entry NV_DATA_STORAGE can be initialized with a new key.

SW_DOWNLOAD_TYPE

Version 2.30 supports the class3 software download. The SW download needs more flash on the target HW.

NV_DATA_STORAGE

Can be set to NV_DATA_SINGLE_BUFFER and NV_DATA_DOUBLE_BUFFER now.

7.1.2.2 Modifications in the startup phase (FF)

There are only minor changes between version 2.2x and 2.30. Please refer to module appl_if.c.

Function appl_start()
The function call appl_get_serial_number() moved to the beginning of the function.

Function appl_get_serial_number()
The calculation of the loop_count variable offers two choices. Disable the unused one.

Function appl_provide_device_ident_to_fdc()
This function is new in version 2.30. The function is used to load specific resource block parameters (Manufacturer ID, Device Type and Min. Cycle Time) to the FF communication stack.

7.1.2.3 Transducer blocks and their methods (FF)

There are only minor changes between the different versions.

Function Appl_update_trn_mode()
One parameter for the function call of ffbl_update_block_alarm() has changed. Please refer to the code of the function.

7.1.2.4 Resource block and its methods (FF)

No changes.

7.2 Differences in PA

7.2.1 New features in version 2.30 compared to version 2.2x (PA)

Version 2.30 has compared to version 2.2x the following new features.

New features	Description
Condensed Status	Condensed status and diagnosis information according to the Amendment 2 to PROFIBUS profile for process control devices V3.01.

Table 9 New features in version 2.30 compared to version 2.2x (PA)

7.2.2 How to migrate from 2.2x to 2.30 (PA)

7.2.2.1 Generation environment (PA)

The initialization for one entry in the file Makeconfig.mak has changed.

NV_DATA_STORAGE
Can be set to NV_DATA_SINGLE_BUFFER and NV_DATA_DOUBLE_BUFFER now.

7.2.2.2 Modification in the device startup phase (PA)

There are only minor changes in case of initialization between version 2.2x and 2.30.

Structure T_FBS_CALL_PDEV_FUNCT
The structure has changed and got the two new components *a_of_usr_prm_handler* and *a_of_check_cfg_handler*. Please refer to fbs_api.h and appl_if.c.

Module appl_diag.c
This module is new. Its used for condensed status and diagnostic calculation.

Function appl_init()
The function initializes the two new function pointers of the structure *T_FBS_CALL_PDEV_FUNCT.*

7.2.2.3 Transducer blocks and their methods (PA)

No changes.

7.2.2.4 Physical block and its methods (PA)

There are some changes in case of condensed status and diagnostic handling.

Function Appl_execute_PHYB()
Code was added to handle the condensed status. Please refer to the code of the function.

Function Appl_usr_prm_handler()
The function is new in version 2.30. Please refer to the code of the function.