GenerateVFD - FF

Functional Description

Version 4.24

Release: 29. November 2011

Softing AG
Richard-Reitzner-Allee 6
D-85540 Haar, Germany
Phone: (++49) 89 45 65 6 - 0
Fax: (++49) 89 45 65 6 - 399

Contents

1 Introduction 4
1.1 Motivation 4
1.2 Concept 4
1.3 Files generated by GenerateVFD 5
1.4 How to generate the Virtual Field Device 6
1.5 Run the program 7
2 Syntax of the script language 9
2.1 How device script is structured 9
2.2 How data structures are defined 10
2.3 General VFD information 12
2.4 How blocks are defined 13
2.4.1 Block information 13
2.4.2 When to use channel information 16
2.4.3 Information about XD_SCALE 17
2.4.4 Information about BLOCK_INSTANCE 17
2.4.5 Information about DEFAULT_CHANNEL 17
2.4.6 How parameters are defined 18
2.4.6.1 Handling of 'mixed' parameters 20
2.4.7 Individual preprocessor statements 21
2.5 Auxiliary files 21
2.6 What the preprocessor does 22
3 Change notes 24

1 Introduction

The model of the VFD (Virtual Field Device) is described in the FF (Fieldbus Foundation) specification FF-870 "Fieldbus Message Specification". VFD is used to describe the externally viewable objects (data types, data structures, parameters, etc.) of the field device. The tool 'GenVFD', engineered by Softing, supports an application developer in creating a function block VFD. It also establishes the frame for a function block application.

1.1 Motivation

The major problem in this respect is to keep the individual data structures (object definitions, dictionaries) consistent as far as changes are concerned: Let's assume that a parameter is to be integrated into the resource block; in this case, you would then have to move all subordinate entries, manually, and adapt all entries in the other data structures (directories, dictionaries).

1.2 Concept

When using GenerateVFD, the files are created from a script and the symbolic output of the DD (Device Description) Tokenizer. The device script contains all the necessary information about the objects in the VFD. GenVFD needs the Tokenizer output (symbol file) to be able to set the correct member and item IDs of the objects. Please make sure to use the same object names in the DD and the script file.

A few objects (e.g., the AP_DIRECTORY and the objects VIEW, ALERT and TREND) are automatically created and therefore must not be considered in the device script. The entries for the block objects are calculated as well.

GenVFD currently supports the following object quantities for a device:

1	VFD
1	Resource block
≤ 37	Function blocks + Transducer blocks (**H1 communication stack**)
≤ 200	Function blocks + Transducer blocks (**HSE communication stack**)
≤ 500	Objects per block (including block object)
1	VIEW_1 object per block
1	VIEW_2 object per block
≤ 16	VIEW_3 objects per block
≤ 16	VIEW_4 objects per block
≤ 31	Objects per one VIEW object
≤ 900	Link objects + Trend objects + Alert objects + Action objects + Domain objects
≤ 25	Max. number of float trends (at manually limitation)
≤ 25	Max. number of discrete trends (at manually limitation)
≤ 25	Max. number of bitstring trends (at manually limitation)
≤ 32	Max. number of MVC elements in one MVC variable list

The following limits apply for the input files of GenVFD:

≤ 1000000	Characters of device script source code
≤ 3000	Characters per script line
≤ 100	Characters object name length, shorted to 32 for FMS-OD
≤ 5000	Items in symbol file
≤ 50	View member IDs in symbol file

1.3 Files generated by GenerateVFD

fbif_dsc.c

This file contains data that describe the device. In detail, these comprise the following entries:

- a VFD – dictionary with pointers to all VFD data
- a structure (‘fbif_block_descr') combining all information required for every block

fbif_id.h

Contains the macros for the MEMBER- and ITEM IDs of all FF objects. These constants are created by the FF - Tokenizer (and stored in the symbol file) and read by GenerateVFD. If the symbol file is not available for some reason, these values must be entered manually. In this case GenerateVFD does not overwrite this file - always delete it explicitly if you want to rebuild it.

< blocktype >.c

In this file are for this blocktype specific data. This file contains following entries:

- the initialization values of every parameter in this blocktype
- description of every parameter in this blocktype
- FMS object descriptions for this blocktype
- view dictionary for this blocktype

< blocktype >.h

Contains the definition of the data structures for the < blocktype > block.

fbif_cfg.h

Contains individual preprocessor statements added in script.

spec_ds.c

Contains the FMS object description of vendor-specific data structures (declared in the script using the keyword 'STRUCT').

spec_ds.h

Contains the type definitions of vendor-specific data structures.

fbif_idx.h

Contains a list of macros for the absolute and relative block parameter indices.

instance.c

Contains data for every block instance:

- initial values of block header
- channel information
- block data

fbif_fct.h

Contains declarations of the interface functions for each block.

fbif_dcl.h

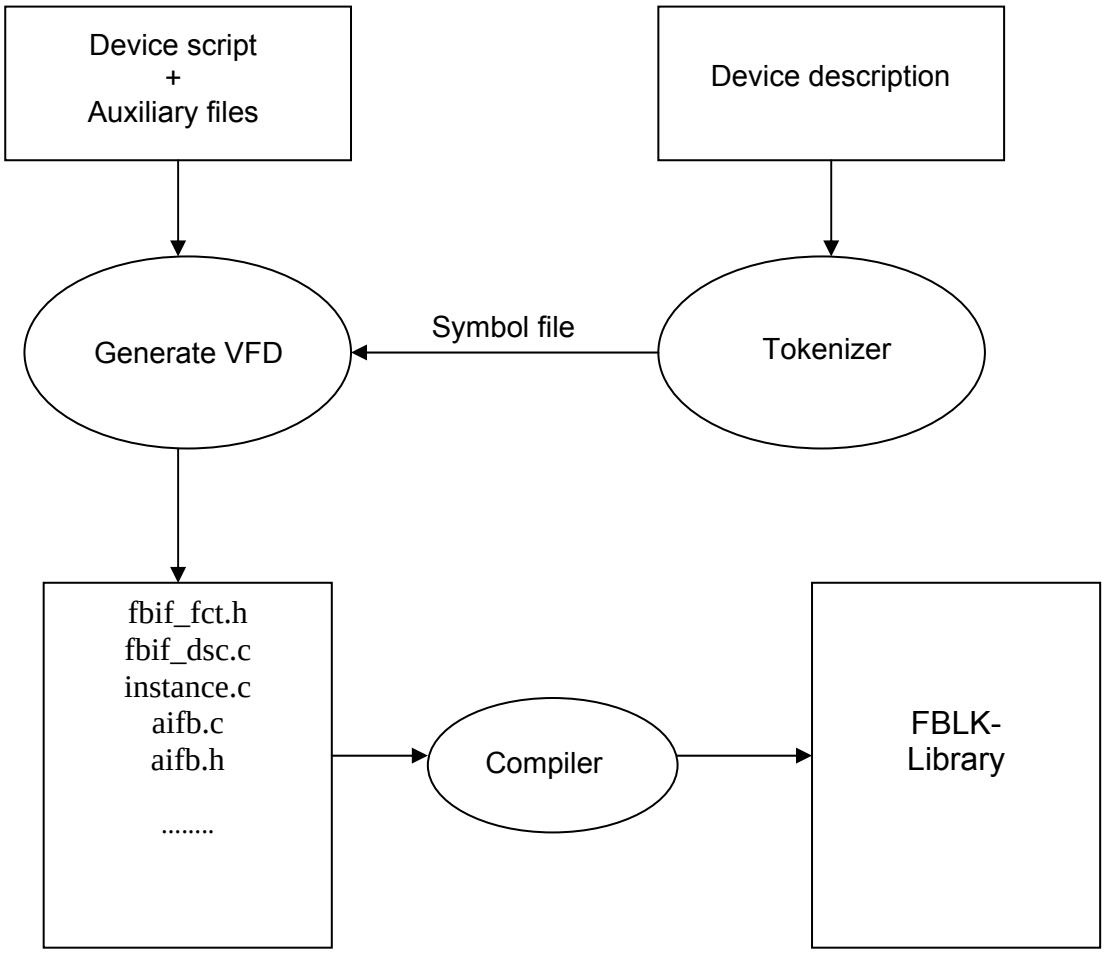
Contains data declarations for each block.

1.4 How to generate the Virtual Field Device

Generation of the function block application can be subdivided into several logical steps:

- Creating the device description
- Translating the DD using the FF Tokenizer
- Creating the device script
- Generate the framework of the application by GenerateVFD
- Compiling the application software (see following figure)

At this point, it might be appropriate to mention that prewritten DDs can be acquired for the standard blocks via the Fieldbus Foundation.



1.5 Run the program

GenVFD is a Win32 console application, controlled by the command line. GenVFD does not require a special installation procedure.

Command Line:

```
genVFD <inc_dir> <src_dir> <sym_file> <input.gw> [options]
```

<inc_dir>	Output directory for generated header files (*.h)
<src_dir>	Output directory for generated source files (*.c)
<sym_file>	Symbol file for the device
<input.gw>	Device definition script file
options:	
-fbif_dict_in_ram	GenVFD will generate a c_fbif_dictionary instead of fbif_dictionary. That makes possible for the application to make an own fbif_dictionary and locate it in RAM instead in ROM.
-hse	With this option GenVFD generates source files for HSE communication stack. Without this option source files for the H1 stack will be generated.
-single_char_no_string	With this option GenVFD initializes block tags with a list of 32 single characters.

Example for H1 communication stack:

```
genVFD ..\inc ..\src 0101.sym test.gw
```

Example for HSE communication stack:

```
genVFD ..\inc ..\src 0202.sym test_hse.gw -hse
```

2 Syntax of the script language

2.1 How device script is structured

The device script contains four sections:

- Script header
- Data structure definitions
- VFD information
- Block definitions

Script Header

Every device script begins with a header line, in which the device name and two identifiers are entered (the identifier codes are insignificant in the current version.)

Example:

```
%Device_AI 0x0001 1
```

	Meaning
`%`	Prefix of the header
`Device_AI`	Device name
`0x0001`	Identifier; not used in the current version
`1`	Identifier; not used in the current version

The header must always be prefixed with a percent character.

Data Structure Definitions

Data structure definitions must be defined after the header. You can declare these using the keyword 'STRUCT'. You may only use the data structures after they have first been declared in this section. See chapter 2.2 How data structures are defined for details.

VFD Information

The third section contains general information about the VFD (e.g.,VFD_Tag, Vendor_name). See chapter 2.3 General VFD information for details.

Block Definitions

The forth section contains the block definitions. See chapter 2.4 How blocks are defined for details.

The script must always end with the keyword 'END'.

2.2 How data structures are defined

The FF specification defines standard data structures that are used for the definition of block parameters. These standard structures are in the file “stdStructs.gw” and are ready to be includes in a device script. Standard structures are identified with the keyword ‘STANDARD_STRUCT’. The names of the standard structures are as defined by the Fieldbus Foundation.

Example:

```
STANDARD_STRUCT FLOAT_S 2 UNSIGNED8,1 FLOATING_POINT,4
```

	Meaning
`STANDARD_STRUCT`	Keyword
`FLOAT_S`	Name of the structure
2	Number of elements in the data structure
`UNSIGNED8,1`	Data type, size of the first element in byte
`FLOATING_POINT,4`	Data type, size of the second element in byte

In addition to the data structures imposed by FF, it is possible to define vendor or device specific data structures. These additional structures are always identified with the keyword 'STRUCT'.
The name of the structure has to be unique to identify the structure in the device script. It can be chosen by the user.
Elements are defined via <Data type, Length>. The length is redundant for a few data types (e.g., FLOAT always has a length of 4), but it must be specified for each element.

Example:

```
STRUCT DATA_STRUCT_87 2 UNSIGNED8,1 INTEGER32,4
```

	Meaning
`STRUCT`	Keyword
`DATA_STRUCT_87`	Name of the structure
2	Number of elements in the data structure
`UNSIGNED8,1`	Data type, size of the first element in byte
`INTEGER32,4`	Data type, size of the second element in byte

There is also another one possibility to define a data structure. With this second description method it is possible to define also the name of each component of the structure. However, it is possible to define a structure in the one or the another way but it is not possible to make one structure definition in a mixed way.

Example:

```
STRUCT MY_STRUCT 2 UNSIGNED8|1|Abc BIT_STRING|2|My_component
```

	Meaning
STRUCT	Keyword
MY_STRUCT	Name of the structure
2	Number of elements in the data structure
UNSIGNED8|1|Abc	Data type, size of the first element in byte, name of the first element
BIT_STRING|2|My_component	Data type, size of the second element in byte, name of the second element

As result of this second syntax for describing a data structure, in the file spec_ds.h GenVFD_FF will generate following C code:

```
typedef struct
  {
    USIGN8 Abc;
    USIGN16 My_component;
  }
MY_STRUCT;
```

The highest number of elements for a read only record or for a record with all writable components is 38. This is a limit given by a PDU size of 126 Bytes. The highest number of elements for a record with mixed access must not be greater than 32.

The following standard data types are supported:

Data Type	Number of Octets	Types in field device software (see base.h and fbap.h)
BOOLEAN	1	BOOL
INTEGER8	1	INT8
INTEGER16	2	INT16
INTEGER32	4	INT32
UNSIGNED8	1	USIGN8
UNSIGNED16	2	USIGN16
UNSIGNED32	4	USIGN32
FLOATING_POINT	4	FLOAT
VISIBLE_STRING	1,2,3,…,32	STRINGV
OCTET_STRING	1,2,3,…,32	OSTRING
DATE	7	DATE_S
TIME_OF_DAY	6	TIME_OF_DAY_S
TIME_DIFFERENCE	6	TIME_DIFF_S
BIT_STRING	1	USIGN8
BIT_STRING	2	USIGN16
BIT_STRING	4	USIGN32
BIT_STRING	8 (for HSE only)	OSTRING [8]
TIME_VALUE	8	TIME_VALUE_S

2.3 General VFD information

This section describes the function block VFD. It has two sub-sections. The first sub-section contains information to identify the function block VFD. The second sub-section may be used to determine the start indices of link, alert and trend objects, the number of links, number of float trend objects, number of discrete trend objects, number of bitstring trend objects, the index of the action object, number of MVC objects, max. number of MVC elements in a MVC object (cannot be higher that 32), start index of MVC objects and start index of MVC variable lists. All keywords in this section are optional. If a keyword is missing the objects are located to default values.

The default values are:

Number_of_links	= 22
Link_Start_index	= 300
Alert_Start_index	= 396
Trend_Start_index	= 340
Action_Object_index	= 399
No_of_MVC_Objects	= 0
Max_No_of_MVC_Elements	= 8
MVC_Start_index	= 380

If the keyword about the start index of the MVC variable lists is missing, it will be set to the first free index after the view objects.

If the keywords for the number of float trend objects, number of discrete trend objects, and number of bitstring trend objects are missing, the number of these objects will be calculated automatically. With this keyword you have possibility to reduce each of this number.

All indices of link, alert, trend, MVC or action objects (=non-block objects) must be either lower or higher than the indices of the block parameters, but always lower than the block view objects and MVC list objects. Also the block object indices always have to be lower that block view objects and MVC list objects.

Allocation of the different object groups (non-block objects, block objects, view object, MVC variable list objects) in the way that the groups would overlap is also not allowed.

Note: If the allocation of the non-block objects will be changed, so it is necessary to change the allocation of *all* non-block objects, as well of the MVC objects, even if they are not used in the software.

Note: All the information's in the second sub-section can be written in a single line, but it can be also written in several lines.

Example:

```
VFD_reference_number=0x1234
VFD_Tag=FB Application
Vendor_name=Softing AG
Model_name=AI Interface
Revision=2.2
```

VFD_INFORMATION No_of_Link_Objects=22 Link_Start_index=300
VFD_INFORMATION No_of_MVC_Objects=1 MVC_Start_index=380
VFD_INFORMATION Alert_Start_index=396

	Meaning
`VFD_reference_number=0x1234`	Keyword=<VFD reference number>
`VFD_Tag=FB Application`	Keyword=<Name>
`Vendor_name=Softing AG`	Keyword=<Name>
`Model_name=AI Interface`	Keyword=<Name>
`Revision=2.1`	Keyword=<Name>

	Meaning
`VFD_INFORMATION`	Keyword
`[No_of_Link_Objects=22]`	Keyword=<number of link objects>
`[Link_Start_index=300]`	Keyword=<Startindex of link objects>
`[Alert_Start_index=396]`	Keyword=<Startindex of alert objects>
`[Trend_Start_index=340]`	Keyword=<Startindex of trend objects>
`[Action_Object_index=399]`	Keyword=<Index of action object>
`[No_of_Trend_Float=20]`	Keyword=<Number of float trend objects>
`[No_of_Trend_Discrete=20]`	Keyword=<Number of discrete trend objects >
`[No_of_Trend_Bitstring=20]`	Keyword=<Number of bitstring trend objects >
`[No_of_MVC_Objects=1]`	Keyword=<Number of MVC objects >
`[Max_No_of_MVC_Elements=8]`	Keyword=<Max. number of MVC elements in a MVC variable list>
`[MVC_Start_index=380]`	Keyword=<Startindex of MVC objects >
`[MVC_List_Start_index=5000]`	Keyword=<Startindex of MVC variable lists >

2.4 How blocks are defined

Each block definition starts with the keyword 'BLOCK_INFORMATION' ends with 'END_BLOCK'. The block description contains information for one block instance. A block is described by:

- Block information
- Channel information
- XD_SCALE information
- BLOCK_INSTANCE information
- Parameter definitions

2.4.1 Block information

The keyword for the definition of block information is referred to as 'BLOCK_INFORMATION'.

Example:

```
BLOCK_INFORMATION Block=RB Tag=RESOURCE BlockType=RESOURCE
ExecutionTime=0 DD_Rev=1 Profile_Rev=1 Start_index=400
StartIndVIEWS=10000
```

	Meaning
`BLOCK_INFORMATION`	Keyword
`Block=RB`	Keyword=<Block specifier>
`Tag=RESOURCE`	Keyword=<Block tag>
`BlockType=RESOURCE`	Keyword=<Block type>
`ExecutionTime=0`	Keyword=<execution time of block in 1/32ms>
`DD_Rev=1`	Keyword=<Revision of device description>
`Profile_Rev=1`	Keyword=<Revision of block profile>
`Start_index=400`	Keyword=<Block start index in OD>
`StartIndVIEWS=10000`	Keyword=<View start index in OD>

All parameters following the block definition are allocated to this block.

Block specifier (internal) consists of two letters identifying the block and an optional consecutive number: FB0 denotes Function Block 0. Resource Block (RB) and Transducer Blocks can also be defined (RB does not contain a number, since only one resource block is allowed per VFD). This name is used to internally identify the block. It may only be specified once per field device.
Possible block specifiers are:

RB	(resource block)
FB	(function block)
TB	(transducer block)

Block tag (external) is used for initializing the corresponding component of the block object (the first object of a block). In the FF network the block appears under this name. The block tag has a maximum length of 32 characters.

BlockType specifies the type of the block (Profile ID) according to the FF device description. The possible block types are defined in file profiles.dat. The file can be enhanced if the device needs additional block types. See chapter 2.5 Auxiliary files for details. The block type has a maximum length of 16 characters.

Possible block types are currently:

	BlockType	**Meaning**
resource block:	RESOURCE	Resource block according FF specification 1.3
	RESOURCE_2	Resource block according FF specification 1.4 and 1.5
	RESOURCE_2_FD	Resource block with Field Diagnostic parameters
	PROG_RES	Programmable Resource Block
function blocks:	AI	Analog Input
	DI	Discrete Input
	ML	Manual loader
	BG	Bias gain
	CS	Control selector

	PD	PD Control
	PID	PID Control
	RA	Ratio
	AO	Analog Output
	DO	Discrete Output
	DC	Device Control
	OS	Output Splitter
	SC	Signal Characterizer
	LL	Lead Lag
	DT	Deadtime
	IT	Integrator (Totalizer)
	SPG	Setpoint Ramp Generator
	IS	Input Selector
	AR	Arithmetic
	TMR	Timer
	AAL	Analog Alarm
	MAI	Multiple Analog Input
	MAO	Multiple Analog Output
	MDI	Multiple Discrete Input
	MDO	Multiple Discrete Output
	PI	Pulse Input
	CAO	Complex Analog Output
	CDO	Complex Discrete Output
	SPID	Step Output PID
	CALCA	Calculate Analog
	CALCD	Calculate Discrete
	DAL	Discrete Alarm
	AHI	Analog Human Interface
	DHI	Discrete Human Interface
	FT	Flow Totalizer
SIF blocks:	SIF_RESB	SIF Resource
	SIF_DO	SIF Discrete Output
	SIF_AI	SIF Analog Input
HSE blocks:	RIO_MBI64	Multiple Binary Input (64)
	RIO_MBO64	Mulitple Binary Ouptut (64)
	RIO_MAI16	Multiple Analog Input (16)
	RIO_MAO16	Multiple Analog Output (16)
transducer blocks:	FCB	Flow calibration basic
	RLCB	Radar level calibration basic
	APVB	Advanced pv basic
	PCB	Pressure calibration basic
	BPVB	Basic pv basic
	TCB	Temperature calibration basic
	TC2S	Temperature with Calibration Two Sensor
	DPVB	Discrete pv basic
	GENERIC_XDCR	Generic Transducer
	xxTB	Other transducer blocks (AITB, DITB, AOTB...)

For manufacturer specific blocks the identifier of BlockType must be equal to block header name (relative block index = 0) in the script and to the block header parameter name prefix at

device description. A BlockType of a manufacturer specific transducer block has to have the suffix “TB”.
Example: if you have a BlockType AITB, the block header parameter name in the device description should be “aitb_character”.

The ExecutionTime of a function block is used by configuration and test tools (e.g., interoperability test) to calculate the macro cycle. It should never be smaller than the actual run-time of the block, but may be larger. Since resource block and transducer blocks are not scheduled their execution time will be set to 0.

The DD_Rev and Profile_Rev shows the revisions of the device description and the block profile.

Start_index is the index of the first block object in the object directory. Start index should always be ≥ 400. Indices ≥ 300 and < 400 will be used for link, alert, trend and action objects.

The start index of the VIEW objects (StartIndVIEWS) must be larger than the recently used index of the block parameter.

The order of describing the blocks in the main script has a significant role. The start indices should have the same order in which the blocks are described in the script. I.e. if you describe the AIFB first in the script and then DIFB, the start index of AIFB should be lower then start index of DIFB.
In an analogous manner the start indices of view objects should have the same order like block descriptions in script, but should be upper than any block start indices.

2.4.2 When to use channel information

Every block instance has a channel information. The channel information is not necessary for every block, but it is necessary for input and output function blocks. All other blocks do not need channel information.
The channel information is used to assign a transducer block to a channel value (parameter 'CHANNEL' in the function block). Multiple declarations are allowed. Depending on the selected channel, one function block may be linked to different transducer blocks. A channel value can appear only once at channel information.

Example:

```
CHANNEL_INFORMATION {1,TB0} {2,TB1} {AO_VAL_CHANNEL,TB2}....
```

	Meaning
`CHANNEL_INFORMATION`	Keyword
`{1,TB0}`	{ <channel number> , <TB specifier> }
`{2,TB1}`	{ <channel number> , <TB specifier> }
`{AO_VAL_CHANNEL,TB2}`	{ <channel number> , <TB specifier> }

As in example, channel number can be also a macro, but it must be defined in the individual preprocessor statements section of this block description (see chapter 2.4.7 Individual preprocessor statements for details).

The channel information is assigned to the current function block and must be defined inside the block definition between BLOCK_INFORMATION and END_BLOCK.

2.4.3 Information about XD_SCALE

Every block instance has a XD_SCALE information. The XD_SCALE information is necessary for analog input and output function blocks. All other blocks does not need XD_SCALE information and it is without meaning.
The XD_SCALE information is used to describe the value obtained from the transducer.

Example:

```
XD_SCALE_INFORMATION {100.0, 0.0, 1342, 2}
```

	Meaning
`XD_SCALE_INFORMATION`	Keyword
`{100.0, 0.0, 1342, 2}`	{ <Eu at 100%>, < Eu at 0%>, <Units Index>, <Decimal Point> }

If no XD_SCALE_INFORMATION defined, the default value { 0.000000, 0.000000, 0, 0 } will be added.

2.4.4 Information about BLOCK_INSTANCE

Every block instance has BLOCK_INSTANCE attribute. This attribute determines if this block is visible or hidden at very first start. It's initial value can be set in script.

Example:

```
BLOCK_INSTANCE HIDDEN
```

	Meaning
`BLOCK_INSTANCE`	Keyword
`HIDDEN`	<HIDDEN or VISIBLE>

If no BLOCK_INSTANCE defined, the default value VISIBLE will be added.

2.4.5 Information about DEFAULT_CHANNEL

This attribute can be used for every block that contains a parameter "CHANNEL". With this attribute you can set any other 2 Byte long default value for this parameter in each block instead of 0.

Example:

```
DEFAULT_CHANNEL 23
```

	Meaning
DEFAULT_CHANNEL	Keyword
23	<channel number>

2.4.6 How parameters are defined

All parameters of a block must be defined between the keywords 'BLOCK_INFORMATION' and 'END_BLOCK'. The index is automatically incremented beginning with Start_index. Each parameter definition is introduced with the keywords STANDARD_PARAM or PARAM.

Example:

```
STANDARD_PARAM ANALOG_INPUT_BLOCK SRW 62 Record BLOCK 13 #0
SubIndexAccess=641
```

	Meaning
STANDARD_PARAM	Keyword for FF standard parameters
ANALOG_INPUT_BLOCK	Parameter name
SRW	Flags
62	Size of the parameter
Record	Object type of the parameter
BLOCK	Data type of the parameter
13	Number of components
#0	#<Initialization value>
SubIndexAccess=641	Write access for records

For standard parameters, parameter names are defined by the FF specifications. After standard parameters, there can be appended device-specific block parameters using keyword PARAM, or standard parameters using keyword STANDARD_PARAM. The name of these device specific parameters is limited to 32 characters.

The parameter names must match the corresponding names in the device description, except that leading underscores (‘_’) are not allowed in the script. Please make sure that the item names in a function block match the corresponding member names in the device description except for using upper or lower case. GenVFD links member and item IDs by comparing the names in the symbol file generated by the Tokenizer.

For standard FF function blocks the first parameter (index 0) should have the same name like the block name in the DD symbol file (standard FF DD of the block).

Example:

*.sym file:

```
     block     __analog_input_block                    0x800201D0
```

Script of the function block:

```
STANDARD_PARAM ANALOG_INPUT_BLOCK SRW 62 Record BLOCK 13 #0
SubIndexAccess=641
```

Flags are used for both the object attribute in the VFD and for the internal database. In the example above, the flags signify the block parameter to be **R**eadable, **W**ritable and **S**tatic.

Permissible flags are:

Readable
Writable
Static (non-volatile and ST_REV is incremented on a write access)
Non-volatile
Output Parameter
Input Parameter
A (read function)
B (write function)
Volatile (ST_REV is incremented but the parameter is not automatically stored in non volatile memory)
X (this parameter will not be stored in memory area of FB application)
Y (default value of this parameter will be called elsewhere)
Z (restart_with_defaults will not change this parameter – valid for static parameters only!)

Flag **Y** should be used only together with **S** or **N** flag. Flag **X** should not be used with **S** or **N** flag.
If the flags **A** and **B** are set, the function block layer informs the application that a read or write access have been occurred. The application may check and possibly refuse the access. For transducer blocks the flags are without meaning. Transducer blocks are part of application and each read and write access on transducer block parameters has to be handled by the application.

The size of the parameter specifies the entire memory space requirement of the object. The maximum permissible size corresponds to the maximum size defined in the FMS specification.

Object type: allowed types are 'Simple', 'Array' and 'Record' according to the FF Specification.

Permissible data types are those defined in the FF specification and all user-defined types.

Number of components refers to records, arrays of visible string and arrays of octet string and is insignificant for all other data types.

Initialization value is entered in file instance.c. Standard parameters are not checked for consistency against the defaults of the FF specification. In records, the initialization values of the individual components must be separated with semicolon (example: `PARAM FIELD_VAL R 5 FLOAT_S 2 #0;0.0`). Values in an array should be separated with comma. Otherwise, for an array of visible string or an array of octet string, the strings should be separated with semicolon (example: `PARAM MY_DATA R 30 Array VISIBLE_STRING 3 #" ";"SECOND ";48,49,50,51,48,49,50,51,52,53)`

SubIndexAccess contains the bite-coded write-access authorization to individual components of a record object. If this parameter is 0, all components are write-authorized. Otherwise the

first (right-aligned) bit corresponds to the first component, the second bit of the second component, etc. If the bits are set, the component is hence write-authorized.

One parameter must be specified in a single line.

Example:
```
STANDARD_PARAM RESOURCE NSRW 62 record F_BLOCK 13 #0
SubIndexAccess=641
```

In this case the GenVFD converts this decimal number (641) to binary (1010000001). This binary number is right-aligned and it means that the first, eight and tenth component of this record can be written. It means further that your write access on this parameter with subindex 1, 8 and 10 will be allowed. Basically the write access on the whole record (subindex 0) will be allowed, but only write permitted parameter will be changed.

In case we have a member of a block which name is different than his DDL item name, it is also possible to assign him this appropriate item ID.
For example if parameter `CHANNEL` in a `xyz_block` should be defined with item `channel_xyz` (extract from a DD source code):

```
BLOCK   ai_block
{
        ...
        PARAMETERS
        {
                ...
                CHANNEL,         channel_xyz;
                ...
        }
}
```

then we can assign the item `channel_xyz` to the parameter `CHANNEL` on this way in a GenVFD script:

```
STANDARD_PARAM CHANNEL|CHANNEL_XYZ SRW 2 Simple UNSIGNED16 1 #0
```

In this case the GenVFD will extract the `CHANNEL_XYZ` item ID from the *.sym file and assign it to the `CHANNEL` parameter. It is important to have **no blanks** between parameter name and his item name, only a vertical bar. The leading underscores in the item name are ignored.

2.4.6.1 Handling of 'mixed' parameters

The field device software does not support storing single components of records. 'Mixed' parameters with static or non-volatile components are stored completely in NV-RAM. The field device software will allocate NV memory if a parameter has got a 'S' or 'N' flag.

The most important 'mixed' parameter is MODE_BLK. The field device software has got special routines to handle write access to MODE_BLK: If Target mode is changed

MODE_BLK will be stored in NV-RAM and the static revision counter will be incremented. If Permitted or Normal mode is changed MODE_BLK will be stored in NV-RAM but the static revision counter will not be incremented. Actual mode is dynamic; a change of the Actual mode does not require an update of the NV-RAM.

In general it is not recommended to use device-specific 'mixed' parameters as those parameters will require special routines which must be implemented as an enhancement of the standard field device software. But there is one supported type of 'mixed' parameters. An example of a supported 'mixed' parameter is ALARM_SUM. The standard parameter ALARM_SUM consists of three dynamic read-only components and one static writeable component. Write access to the static writeable component is handled by standard routines, while write access to the dynamic read-only components is ignored (if the whole parameter is written) or rejected (if a single read-only component should be written).

2.4.7 Individual preprocessor statements

With the ‘%’ sign individual preprocessor statements added in script can be included to file fbif_cfg.h. It allows generation of code without the necessity for additional modifications in the file headers.

Example:

```
%#include "my_defines.h"
%#define MY_VALUE     3
```

This statements in script adds the lines

```
#include "my_defines.h"
#define MY_VALUE     3
```

into the file fbif_cfg.h.

This statement is only possible within a block declaration and causes an error message in all other cases.

2.5 Auxiliary files

The block profiles and view objects are defined in two support files called ‘profiles.dat’ and ‘views.dat’. The view objects for standard blocks are predefined and stored in file ‘std_views.dat’ which is included in ‘views.dat’.

Profiles.dat

The profile numbers correspond to the component ‘profile’ in the block object and identifies the block type. All profile numbers are defined by the Fieldbus Foundation and must not be changed.
If new block types are used their profile numbers have to be added to this file. The block type names used in the device description (e.g. for naming view objects) must be a substring of the name used in profiles.dat and views.dat.
The profile numbers are not block specific but block type specific (just one entry per used block type has to be included).

Example:

AOFB 0x102

Where AOFB is the BlockType (see chapter 2.4.1 Block information) and 0x102 the corresponding profile number according to FF specification.

Views.dat

The usage of ‘views.dat’ is similar to ‘profiles.dat’. For each used block type there must be one view description in this file. The block type is defined by using the keyword ‘Block’ followed by the block type specific name (see example below). Each block has 4 mandatory view objects called VIEW_1 to ..._4. The keywords VIEW_<n> are followed by the relative object indices of the objects which are intended to be compiled to the VIEW_<n> object (the view objects are variable lists). Notice: a minimum set of objects have to be integrated to each view object depending on the block type and index of the view object. One can add objects to this set without disturbing the interoperability of the block but one must not remove objects which belong to the minimum set.

Example:

```
Block AOFB
VIEW_1 1,5,6,7,8,9,16,17
VIEW_2 1,11,12,13,20,21
VIEW_3 1,5,6,7,8,9,16,17,25,26,28
VIEW_4 1,3,4,14,15,18,19,22,23,24,27
```

The block names defined in this files have to be used in the block definitions of the device scripts (*.gw - Files)

Example:

```
BLOCK_INFORMATION ........  BlockType=AOFB  .......
```

2.6 What the preprocessor does

The device script is pre-processed. Comments can be inserted at any location, corresponding to standard C-syntax.

Example:

```
//-------------------------------------------
// Function Block 0   (AI)
//-------------------------------------------
```

Similarly, one can use the #include - and the #define mechanism in order to import completed code sections or to define macros.

Example:

```
#include "aifb.gw"
```

Search is made in the current directory for the file 'aifb.gw', and the code is inserted from this file at the current position.

Example:

```
#define START_INDEX_RES_BLK 400

BLOCK_INFORMATION Block=RB Tag=RESOURCE BlockType=RESOURCE_2
ExecutionTime=0 DD_Rev=1 Profile_Rev=1
Start_index=START_INDEX_RES_BLK ...
```

These lines result after preprocessing in #

```
BLOCK_INFORMATION    Block=RB    Tag=RESOURCE    BlockType=RESOURCE_2
ExecutionTime=0 DD_Rev=1 Profile_Rev=1 Start_index=400 …
```

3 Change notes

Changes in version 3.00:

Adapted to version 2.10 of Softing's Foundation Fieldbus communication software:

(i) New interface to communication software
(ii) Generated files have new names
(iii) New keywords added to script
(iv) Macros are now also allowed at channel information

Changes in version 3.01:

(i) Support of array of visible string and array of octet string added
(ii) Support of `DEFAULT_CHANNEL` added
(iii) Header file fbif_cfg.h will be included to the fbif_dsc.c file, so that application-specific macros can be applied within fbif_dsc.c
(iv) The generated sources are adapted for IAR Compiler 2.11a
(v) Correction of generating the fbif_dictionary for the components VFD_Tag, Vendor_name, Model_name and Revision. The length of this strings (32 Bytes) in the script will be checked
(vi) Components offset will be now calculated by compiler with the macro "offset"

Changes in version 3.02:

(i) The generated sources includes the segment definition header files
(ii) The function block data will be stored now in the FBIF_DATA segment
(iii) Permitted length of a source file (script file, dd symbol file, etc.) increased from 200 kByte to 250 kByte
(iv) Number of possible trend objects changed from 100 to 200
(v) For the 2.xx IAR Compiler: The suppressing of the warning Pe991 (extra braces are non standard) will be added to the generated sources
(vi) For the 2.xx IAR Compiler: NO_INIT macro added by the definition of variables

Changes in version 3.03:

(i) Now it is possible to make more settings in the VFD Information section
(ii) All settings in the second sub-section of the VFD Information section are now optional. That makes the previous GenVFD script files compatible to the new ones
(iii) The `fbif_dictionary` in the generated file fbif_dsc.c has some more components now. Consequential the generated sources are no more compatible to the previous stack versions <2.11
(iv) Small plausibility check of the input data at the VFD Information section added
(v) If no UPDATE_EVT or BLOCK_ALM objects found in a block, GenVFD will generate a warning message instead of an error message
(vi) New command line option `fbif_dict_in_ram` added
(vii) Now there is permitted to use 12 instead of 10 `VIEW_3` and 12 `VIEW_4` objects
(viii) If no `VIEW_X` item IDs found in a *.sym file, the GenVFD will generate a warning instead of error. The IDs are set to 0x00

Changes in version 3.04:

(i) GenVFD generates a warning if a member ID or a item ID can not be found in the *.sym file
(ii) GenVFD generates an error message if the number of elements in a VIEW object is greater than 31
(iii) An additional consistency check on VIEWs is implemented: if the relative index of a view element is greater than the number of parameters of the block, GenVFD generates an error message
(iv) GenVFD generates a warning message if the length of the parameters in a VIEW is greater than 123 bytes

Changes in version 3.05:

(i) If using the BLOCK_INSTANCE attribute with value HIDDEN, the GenVFD generates now BLK_INACTIVE macro instead of BLK_HIDDEN macro in the block description. The macro BLK_HIDDEN does not exist in the stack software.
(ii) Check of overlapping of block indices added.
(iii) Correction of generating record descriptions at spec_ds.c. It affect only vendor-specific data structures with components of type TIME_VALUE.
(iv) At optimization level “Maximize Speed” the Microsoft Visual Studio C++ 6.0 Compiler seems to make a compiler defect. That’s the reason why now the optimization level is “default”.

Changes in version 3.10:

(i) New options added:
- symbolic_block_id
- device_type=XXXXDD
(ii) Comments with parameter name for each parameter in the record with initial values added.
(iii) Check for the keyword END at the end of main script added.
(iv) Bugfix for parameters that have 32 characters added.

Changes in version 3.20:

(i) Option - symbolic_block_id is no more necessary, because GenVFD now always generates the symbolic block IDs.
(ii) In case when the length of the parameters in a VIEW object is greater than 123 bytes GenVFD will expand the warning message with the length of this VIEW object.
(iii) GenVFD supports now new additional syntax for parameter definitions. With this new syntax it is possible to define a parameter with separated member name for member ID, and item name for item ID.

Changes in version 4.00:

(i) Support of HSE communication stack added.
(ii) Possibility to set the number of float trend objects, number of discrete trend objects, number of bitstring trend objects manually (reducing it).
(iii) Bugfix: When number of tags and corresponding length statement at definition of a record are not equal, the GenVFD will generate an ERROR message, but now it also stops generating incorrect sources.
(iv) Keyword "No_of_Link_Objekts" corrected to "No_of_Link_Objects". The old keyword will be still accepted because of compability with old scripts.

(v) Flags X, Y and Z are now supported
(vi) Possibility to add hex value for DEFAULT_CHANNEL added.

Changes in version 4.10:

(i) The GenVFD now detects invalid data types used in a STRUCT definition.
(ii) FMS object description in case of array of visible strings and array of octet strings fixed.
(iii) Check if number of components in a script is >0 added.

Changes in version 4.20:

(i) Warning for the case the block tag is longer than 16 bytes added.
(ii) Definition of variable object_dict_ext_ds corrected, because of an IAR Complier V3.20 error message.
(iii) The length of array elements in the FMS object description are now calculated correctly.
(iv) Correction of allocating the item ID in case of differences between member name and item name.

Changes in version 4.21:

(i) Support of Multiple Variable Containers (MVCs) added. With this change the new GenVFD_FF version is no more compatible to the older stack versions (<V2.20) that have no MVCs.
(ii) Structure definitions of block NV parameter and block STATIC parameter added to the block header files.
(iii) The eep length at fbif_block_descr record replaced by a length of NV data and a length of STATIC data.
(iv) Support of a background function for every block added to the fbif_block_descr record.
(v) Support of the SIS blocks added.
(vi) First 4.21 versions of GenVFD_FF didn't generate the right length of NV data in a block if the block type exist more than once in the project. This is corrected since version 4.21.0.05.

Changes in Version 4.22:

(i) Change in this documentation: description of the highest number of elements for a record with all writable components corrected.
(ii) Now is no more possible to define an array of records. The existing check was corrected. In this case an ERROR message appears.
(iii) In case of a definition of a record of records, the ERROR message text is now more meaningful.
(iv) In case of using the option "-device_type=" the generating of block-type header files is corrected.
(v) Check if all 4 VIEW objects are correct defined added. For the case they are not correct defined the GenVFD_FF prints now an ERROR message.

Changes in Version 4.23:

(i) GenVFD_FF supports now generating manufacturer specific data structures with meaningful names for each component. This is possible with the new syntax which is now also described in this document.

(ii) Check of overlapping indices of non-block objects corrected.
(iii) New object overlapping checks added: check of overlapping the view objects, check of overlapping between the non-block, block, view and MVC list objects.
(iv) GenVFD_FF now checks if view and the MVC list objects are located after block and non-block objects.
(v) Bug for the case of a block with more than 256 parameters corrected.
(vi) The use of MVC objects is now also documented.
(vii) The input of VFD_INFORMATION is now corrected – now it is possible to use all 12 keywords instead of only 8.
(viii) The input of VFD_INFORMATIONS is now also possible in multiple lines.
(ix) Parameters with X Flag don't need to have a default value in script any more.
(x) An ERROR message will be generated if a read-only parameter of type record has the SubIndexAccess attribute which is ≠ 0.
(xi) Description for the Z flag adjusted in the documentation.

Changes in Version 4.24:

(i) Because of change of some block identifier in the DD library since the V3.4, now instead of the SIS blocks the SIF blocks will be supported.
(ii) Support of additional blocks added, so GenVFD_FF now supports all the blocks which are defined in the DD library V3.41. All supported blocks are also described in chapter 2.4.1 Block information.
(iii) Now the max. number of VIEW_3 and VIEW_4 objects per block which the GenVFD_FF supports is increased from 12 to 16.
(iv) Support of BlockType RESOURCE_2_FD added.
(v) Wrong check of overlapping indices corrected (defect #583). This check is wrong in case:
    (1). there are no MVC objects and
    (2). the non-block objects are not on the begin of the address space
(vi) Support of arrays of bitstrings corrected (defect #588).
(vii) New command line option –single_char_no_string added.
(viii) Support of bitstring with length of 8 octets (HSE only) added (defect #38 in FF HSE Device Testtrack database)
(ix) Support of the following blocks added (defect #39 in FF HSE Device Testtrack database):
    - __RIO_MBI64_BLOCK (0x014D)
    - __RIO_MBO64_BLOCK (0x014E)
    - __RIO_MAI16_BLOCK (0x014F)
    - __RIO_MAO16_BLOCK (0x0150)
(x) New section in this document: 2.4.6.1 Handling of 'mixed' parameters