**ADDING AND MODIFYING FF CUSTOM PARAMETERS**

*This document is intended to provide guidance for adding and modifying parameters of the Transducer Block of the SVI-II Foundation Fieldbus device.*

*In this document we intentionally avoid using the word “variable” in the meaning of the FF parameter. This is because the word VARIABLE as it will be shown below has a very important meaning as a constituent of PARAMETER. FF community uses the object paradigm therefore the term “block object” and the term “block parameter” (shortened to “parameter”) are used interchangeably.*

*This document implies familiarity with the code base structure and interconnection between DD and Firmware along with procedures and rules for building the of DD and FF firmware. These prerequisite topics are detailed in the related document.*

**Overview**

Generating FF firmware is a complex process which is comprised of manual and automated procedures. The diagram below presents a bird’s eye view of FW build process and reflects involved stages.

**DDL File (text)**

**Tokenizer**

**GenVFD**

**Binary DD**

**GenVFD**

**IAR Build**

**GW Script (text)**

**VFD definitions**

**.C files**

**IAR Build**

**Manually Defined Source**

**.C file**

***DCS Function***

**Files to Modify**

The engineer involved in defining new FF parameters must deal with 3 files as shown in the table below.

|  |  |  |  |
| --- | --- | --- | --- |
| **File Type** | **Extension** | **Present Implementation** | **Belongs to:** |
| DDL file | .ddl | svi\_positioner.ddl | DD side |
| Firmware DD script | .gw | SVI\_Positioner\_APP.gw | FF firmware script |
| FW source code | .c | appl\_ptb.c | FF firmware |

The flow of work required to modify or create new parameter may be illustrated by the following diagram:

**Types of Parameters by Read/Write Methods**

Modify DDL file

svi\_positioner.ddl

Modify GW Script

SVI\_Positioner\_APP.gw

Modify Source File

appl\_ptb.c

Build Entire FW

Load to Device

With IAR

From the execution point of view there exist 3 types of parameters:

1. Cyclic high priority
2. Cyclic low priority
3. Non-cyclic

Cyclic high and low priority parameters are updated by the FW periodically. High priority parameters are addressed more frequently than low priority parameters.

Non-cyclic parameters are updated and/or read only when requested explicitly by the control tool (like NI Configurator).

Note on NI Configurator: While being read values of cyclic parameters are displayed in the parameter windows automatically without any user actions. When a parameter is written by a user the NI Configurator communicates the new value to the FF firmware only in response to the user action – hitting a “write” button.

Important result: above feature has an important effect – parameter which may be both written and read if they are made cyclic display erratic behavior. Being written by a user they sporadically get or get not written to the device. It happens because being written to the application’s window the value may overwritten in this window by incoming read, then transmitted to the device as if it is a new value. Sometimes however the user may be fast enough to hit the “write” button before the window is overwritten – than the new value is written to the device.

The above consideration proves that read/write parameters shall never be made cyclic when NI Configurator is intended as the Control Tool.

**Structural Types of Parameters**

In respect to the parameter organization there should be distinguished only two types of parameters:

1. Simple, consisting of a single VARIABLE of any scalar type
2. Structure, consisting of a single RECORD, which in turn may consist of arbitrary number of VARIABLES.

This organization is reflected with some differences by DDL files and C source code. Our purpose is to define and modify parameters in the FF program we will study how these definitions and modifications are spread among all files comprising a program.

**DDL File**

The detailed description of DDL files is provided in the Foundation Fieldbus manual FF-900: “Device Description Language”. FF-900 is a necessary reference for complex work on DDL files, including adding and modifying very complex parameters. For simple cases this document should suffice.

For immediate reference see the DDL file for the SVI FF device: <svi\_positioner.ddl>.

All standard DDLs are included in the code base and are searchable – search items preceded by double underscore in the DDL files of the code base.

In this document we will overview simple and most frequently encountered cases.

Every work on adding/modifying parameters must start with DDL file.

Variables must be described with the set of attribute, some of which are mandatory, others are optional.

Example:

BLOCK PTB

{

CHARACTERISTICS ptb\_character ;

LABEL LBL\_POSITIONER\_TRANSDUCER\_BLOCK ;

HELP HLP\_POSITIONER\_TRANSDUCER\_BLOCK ;

PARAMETERS

{

/\* standard parameters \*/

ST\_REV, \_\_st\_rev ;

TAG\_DESC, \_\_tag\_desc ;

FINAL\_VALUE, \_\_final\_value;

FINAL\_POSITION\_VALUE, \_\_final\_position\_value;

.....other standard parameters....

...................................

/\* non-standard parameters \*/

/\* most likely here we are inserting new parameters \*/

CONTROL\_PARAMETERS, svi\_control\_parameters\_record;

FIND\_STOPS, svi\_find\_stops;

SUPPLY\_PRESSURE, svi\_supply\_pressure;

ACTUATOR\_PRESSURE, svi\_actuator\_pressure;

**RAMP\_RATE, ramp\_rate;**

**SLEEP, little\_nap;**

.....other non-standard parameters....

......................................

}

....other block components: vies, etc......

PARAMETER\_LISTS

{

VIEW\_1, ptb\_view\_1 ;

VIEW\_2, ptb\_view\_2 ;

VIEW\_3, ptb\_view\_3 ;

VIEW\_4, ptb\_view\_4 ;

}

..............................................

..............................................

} /\* end block definition \*/

DDL files use regular C-language preprocessor. It accepts #define statements, inclusions of <.h> files and standard C comment delimiters (no C++ comments). In addition DDL supports importing other DDL files to reuse standard and 3rd party definitions.

Block parameter definitions consist of 2 mandatory components – Name and DD Item – and 2 optional – description string and help string. Strings may be replaced by their defined names, with definitions elsewhere in the same file or included file.

In the PARAMETER definition DD Item may be:

- VARIABLE name defined in the file or in an imported file;

- RECORD name

Double underscore “\_\_” before the DD Item mean that it is a standard DD Item, defined in one of the standard DDL files which must be included in each custom DDL.

Note that standard DD Items may be redefined with the special word REDEFINE by the custom DDL author so that part of it is modified and part remains unchanged.

Using a parameter of the standard type is trivial – we only need to include the DDL file where standard DD Item is defined. Name may be arbitrary.Custom parameters need to have their DD Items defined prior to the block definition.

More simple case is a scalar parameter presenting only a single variable (as mentioned above).

VARIABLE definition consists of its name and attribute as in the following example:

VARIABLE ramp\_rate

{

LABEL LBL\_RAMP\_RATE ;

HELP HLP\_RAMP\_RATE ;

CLASS CONTAINED ;

TYPE FLOAT ;

HANDLING READ & WRITE ;

}

ramp\_rate is the VARIABLE name, which will be used as a DD Item in the PARAMETER definition. The attributes are in curly brackets.

Simple single-value parameters are seldom used in DDL definitions. Normally at least a single-byte status must be used along with the numeric value. To handle multi-variable parameters DD use RECORDs. They are similar to C-language structs and consist of two or more VARIABLEs as in the example below:

RECORD svi\_supply\_pressure

{

LABEL "svi\_supply\_pressure";

HELP "svi supply pressure";

MEMBERS

{

STATUS, \_\_status\_contained\_nd ;

VALUE, \_\_float\_contained\_nd ;

}

}

This is a relatively simple example because it does not require to define VARIABLEs comprising the RECORD. These variables STATUS and VALUE are standard types

<\_\_status\_contained\_nd > and <\_\_float\_contained\_nd > defined in the standard included DDL “std\_parm.ddl”. As an example this definition from the standard file is quoted here:

VARIABLE \_\_float\_contained\_nd

{

LABEL [value\_label] ;

HELP [float\_contained\_help] ;

CLASS CONTAINED & OPERATE ;

TYPE FLOAT ;

HANDLING READ & WRITE ;

/\* RESPONSE\_CODES xxx ; \*/

}

It is important to notice that such or similar RECORD is used most often in DDL definitions, because most parameters use some value and its status. Value is not necessary floating point. For similar definitions of non-floatingvariables see “**std\_parm.ddl**” or search the DDLs included in the code base.

Finally we will consider a complex case when a RECORD needs full definition (from scratch).

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

little\_nap

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

\*/

VARIABLE time

{

LABEL "Sleep length" ;

HELP "Duration of snooze during this presentation in min." ;

CLASS CONTAINED ;

TYPE FLOAT ;

HANDLING READ & WRITE ;

}

VARIABLE uninterrupted

{

LABEL "Is uninterrupted" ;

HELP "True if your sleep uninterrupted all the way through" ;

CLASS CONTAINED ;

TYPE BOOLEAN ;

HANDLING READ & WRITE ;

}

RECORD little\_nap

{

LABEL "Nap quality" ;

HELP "Quality of your sleep during this meeting" ;

MEMBERS

{

LENGTH, time ;

QUALITY, uninterrupted;

}

}

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

The additional topic important for defining DDLs is definitions of VIEW lists.

FF Specification prescribes 4 views for each block which allows grouping of parameters which need to be displayed.

To make a very brief description of this topic we will mention that parameters in VIEWs may be selected by their dynamic properties, which means how they have to be updated and displayed by an application.

In order to associate parameters with different vies FF uses PARAMETER\_LISTs as presented in our sample block above. Mandatory names are VIEW\_1, VIEW\_2, VIEW\_3 and VIEW\_4. The DD Items are defined as VARIABLE\_LISTs like in the example below:

VARIABLE\_LIST ptb\_view\_1

{

LABEL LBL\_PTB\_VIEW\_1 ;

HELP HLP\_PTB\_VIEW\_1 ;

MEMBERS

{

VL\_ST\_REV, PARAM.ST\_REV ;

VL\_MODE\_BLK, PARAM.MODE\_BLK ;

VL\_BLOCK\_ERR, PARAM.BLOCK\_ERR ;

..............................

}

}

In this example member names start with VL\_ and the parameters defined elsewhere and used in the block description are used in the conjunction with the word PARAM separated by a dot.

The above text was predominantly concentrated on defining new parameters. Modifying existing variables – especially standard parameters is simple. The parameter first be imported, then some of its component must be redefined with the word REDEFINE as dictated by block definition. The snipped from the file “svi\_positioner.ddl” is shown here as an illustration. Examples of other cases may be found in the same file.

IMPORT MANUFACTURER \_\_FF,

DEVICE\_TYPE \_\_STD\_PARM,

DEVICE\_REVISION SFT\_STD\_PARM\_dev\_rev,

DD\_REVISION SFT\_STD\_PARM\_dd\_rev

{

EVERYTHING ;

REDEFINITIONS

{

REDEFINE VARIABLE \_\_manufac\_id

{

LABEL [manufac\_id\_label] ;

HELP [manufac\_id\_help] ;

CLASS CONTAINED ;

TYPE ENUMERATED (4)

{

{0x445644, "|en|Dresser Valve Division", [mfr\_id\_help]}

}

CONSTANT\_UNIT [blank] ;

HANDLING READ ;

}

REDEFINE VARIABLE \_\_dev\_type

……………………………………………………………………….

This snippet concludes the description of the modifications to the DDL file. The next important file to change is the script file <.GW>, which belongs to the FF side firmware source and must be modified in conjunction with the DDL.

**Firmware DD Script**

Note: *VFD – “Virtual Field Device”, FF term for software representation of the physical device*.

In the present code base the GW file of interest is “**SVI\_Positioner\_APP.gw**”, which along with all other scripts resides in the folder: <**….\ FD-SW\target\appl\fbif\script**>. The purpose of this file is to define the structure of VFD blocks (including Transducer Block). VFD blocks defined in GW scripts are used for generating code which presents given blocks to an application.

Definitions of GW scripts are simple. One needs to list the parameters which comprise the VFD block and with each parameter provide the following attributes:

* Name **same as in DDL file**
* Flags (such as Read/Write)
* Length in bytes
* Structure or simple variable type (examples: Simple INTEGER32; Record FLOAT\_S)
* Number of components (such as number of Record members)
* Initial values of all components (denoted with the pound sign # and delimited by semicolon)
* SubIndexAccess, bit-coded write-access authorization for individual components. Zero for write-authorization for all components.

//--------------------------------------------

// Transducer Block 0

// Analog Input Transducer Block

//--------------------------------------------

BLOCK\_INFORMATION Block=TB0 Tag=POSITIONER\_TB BlockType=PTB ExecutionTime=0 DD\_Rev=1 Profile\_Rev=1 Start\_index=START\_INDEX\_TR\_BLK0 StartIndVIEWS=START\_INDEX\_VIEWS\_TBLK0 NrVIEW3=1 NrVIEW4=1

#include "positioner\_tb.gw"

// index = 14

STANDARD\_PARAM FINAL\_VALUE RW 5 Record FLOAT\_S 2 #0;0.0 SubIndexAccess=0

// index = 15

STANDARD\_PARAM FINAL\_POSITION\_VALUE RW 5 Record FLOAT\_S 2 #0;0.0 SubIndexAccess=0

.........................................

// index = 18

PARAM FIND\_STOPS RW 4 Simple INTEGER32 1 #0

.........................................

// index = 20

PARAM SUPPLY\_PRESSURE R 5 Record FLOAT\_S 2 #0;0.0 SubIndexAccess=0

// index = 21

PARAM ACTUATOR\_PRESSURE R 5 Record FLOAT\_S 2 #0;0.0 SubIndexAccess=0

END\_BLOCK

For reference and further examples, please, see files in the folder <….\ FD-SW\target\appl\fbif\script >

**Firmware Source Code Modification**

The last component to be changed is the file “**appl\_ptb.c**” in the folder <**...\FD-SW\target\appl\fdev\src**>

There are 4 spots in this file which need to be modified:

1. The function < **Appl\_handle\_HART\_cmd()** >;
2. The function < **Write\_handler\_PTB()** >;
3. Two lists denoted as **/\* Cyclic commands \*/** and **/\* Regular commands \*/** at the beginning of the file;
4. Definition segment at the top of the file where command numbers and macros like CMD\_...\_FINAL\_VALUE\_OFFSET are “pound-defined”.

Most of work is done in the function Appl\_handle\_HART\_cmd(). The function presents a long “switch” statement, where each HART command is presented by a “case”. Adding or modifying a block parameter is reduced to adding or modifying a “case” as in the following example:

case CMD\_154\_READ\_SETPOINT:

{

if ( access == RECEIVE )

{

/\* read FINAL\_VALUE from HART \*/

READ\_HART\_CMD(CMD\_154\_FINAL\_VALUE\_OFFSET, p\_PTB->final\_value.value , FLOAT);

p\_PTB->final\_value.status = SQ\_GOOD\_CAS | SUB\_NON\_SPECIFIC;

}

else if ( access == SEND )

{

\*send\_length = HART\_READ\_CMD\_REQ\_LEN ;

}

break;

} /\* end case CMD\_154\_READ\_SETPOINT \*/

case CMD\_155\_WRITE\_SETPOINT:

{

if ( access == RECEIVE )

{

/\* read FINAL\_VALUE from HART \*/

/\* there is no data to read \*/

}

else if ( access == SEND )

{

WRITE\_HART\_CMD(CMD\_155\_FINAL\_VALUE\_OFFSET, SPSubCmd, USIGN8);

WRITE\_HART\_CMD(CMD\_155\_FINAL\_VALUE\_OFFSET + 1, p\_PTB->final\_value.value, FLOAT);

WRITE\_HART\_CMD(CMD\_155\_FINAL\_VALUE\_OFFSET + 5, SPmode, USIGN8);

\*send\_length = CMD\_155\_REQUEST\_LENGTH ;

}

break;

} /\* end case CMD\_155\_WRITE\_SETPOINT \*/

The meaning of the variables “access” is obvious.

“p\_PTB” is a pointer to the VFD representation of the block (in our case Position Transducer Block).

Note: *this block is presented by NI Configurator as a list of parameters, with which we, users, work via the application.*

The rest of the “case” is the number of macros of types WRITE\_HART\_CMD() and READ\_HART\_CMD()

READ macros read the HART receive buffer at the offset controlled by macros CMD\_...\_FINAL\_VALUE\_OFFET into the elements of the “p\_PTB” structure. The 3rd argument of the macro, specifying the type, controls the length of the read variable.

WRITE macros read the elements of the “p\_PTB” structure and write them to the HART send buffer at the offset specified by macros CMD\_...\_FINAL\_VALUE\_OFFET. Again the length of the buffer read is specified by the 3rd argument.

Important Notes:

1. Macros CMD\_...\_FINAL\_VALUE\_OFFET have exactly the same values as offsets of corresponding parameters of HART commands from AP FW.
2. Elements of the VFD structure (e.g. p\_PTB) used in macros WRITE and READ must be exactly same as names of the parameters defined in GW script, but transformed to low case.

**/\* Cyclic commands \*/** and **/\* Regular commands \*/** segments of the code denote respectively high priority commands that are called cyclically most often and low priority commands also called cyclically but at longer periods.

Commands which must be called periodically should be merely placed in these lists. The programmer should not call them explicitly – the program does it.

Note: The term “Regular commands” is misleading – it may seem to denote non-cyclic commands, however it does denote cyclic although less frequent commands.

Non-cyclical parameters are those which updated only by user request (via the control tool). Updates are made by non-periodic HART commands calls to which are made explicitly via the function appl\_send\_acyc\_HART\_cmd() which takes the index of the required parameter as an argument. The function appl\_send\_acyc\_HART\_cmd()is placed in the code inside the function Write\_handler\_PTB()**.** The same handler function performs mode check (and other necessary checks) based on the parameter index pound-defined in the form: REL\_IDX\_PTB\_[param-name] inside the automatically generated file <fbif\_idx.h>.

For detailed references and further examples, please, see the file ...\FD-SW\target\appl\fdev\src\appl\_ptb.c

----------------------- End of Document -----------------------