Smart solutions for comfort and safety

saia-burgess
Smart solutions for comfort and safety

File System User Interface Document

SFB (xx7) Interface Description

Version:	1.6
Date:	May 3rd, 2006
Status:	Released
Classification:	Public

Contributors:

Names	**Company**
Schneeberger Francis	SBC
Kafandaris Konstantinos	SBC

Revision History:

Version	**Primary Authors**	**Description of Version**	**Date Approved**
Draft	F. Schneeberger	Draft version	March 2005
1.0	F. Schneeberger	Initial Version	19 Mai 2005
1.1	F. Schneeberger	Added SFB 461 / SFB 462 / SFB 463	9 June 2005
1.2	F. Schneeberger	Finalisation GroupID and GroupAccess added	2 September 2005
1.3	F. Schneeberger A. Roethlin	§2.1 Parameter description for device name §2.2 Error code updated §3.12 : Function description has been rewritten §3.13 : Function description has been rewritten and `Err` parameter has been replaced with `Done` parameter §3.14 : USize and FSize have been exchanged.	March 7th, 2006
1.4	F. Schneeberger	§3.12 : Format can be done with a different block size	March 17th, 2006
1.5	F.Schneeberger	FB 464 (device information) added FB 465 (device status) added	March 23rd, 2006
1.6	F.Schneeberger	GroupID definition reviewed	May 3rd, 2006

Table of Contents

1 Introduction

1.1 Purpose of this document

The File System FB Library allows accessing the different file systems created on a PCD by specific function blocks.

The library includes the following file functionalities:

- creation (file or directory)
- opening a file
- closing a file
- deletion (file or directory)
- reading a file
- writing an open file
- seeking into an open file
- Opening / reading / closing in one call
- Opening (creating if not exist) / writing at end of file / closing in one call
- Formatting a flash device
- Compressing a flash device
- Getting device information

1.2 The File System

The File System is an internal data storage which can be accessed through this API.

The data is located either on a flash device or on a SRAM / SDRAM device and remains passive until a user program needs to update, add or work with file information.

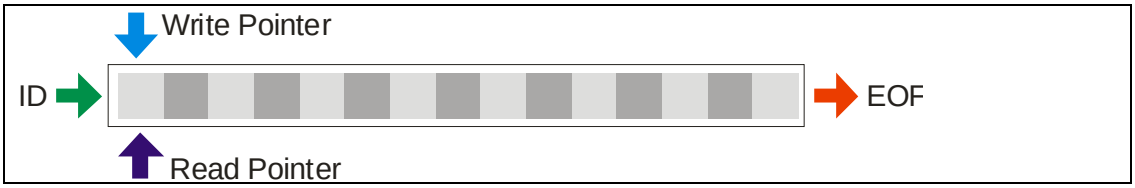
The essential file system usage steps are:

1. create, open a file by means of its name
2. optionally seek a position in the file
3. write data from the PLC context to the file
4. read data to the PLC context from the file
5. close the file

Additionally it is possible to delete a file, a directory (and all files contained in it) or inquire its size.

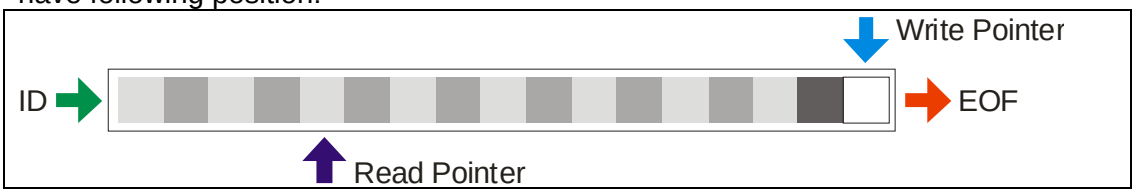
By opening or creating a file you receive an identifier. The identifier is hooked with two pointers:

- a read position pointer
- a write position pointer



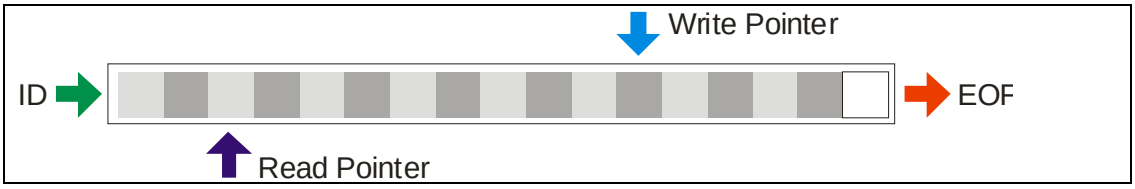
The pointers are modified by calling read, write or the seek function.

When 4 bytes of data are read and one byte is written at the end of the file, the pointers will have following position:



The seek function modifies the read/write pointers relative to current pointer position.

In this example -2 read bytes and -5 write bytes were independent sought.



2 File System FB Library

2.1 Parameters

Name	Used	Description

FileName	FCREATE
	DCREATE
	FOPEN
	DELETE
	SWRITE
	SREAD
		The file name can be passed as absolute pathname, e.g. M1_FLASH:/Report.txt. The file name may contain only alphanumeric characters (without SPACE) and ".". A filename or directory name can not exceed 24 characters, including extension and the total length of a passed absolute filename shall not exceed 64 characters.
		Valid device names are:
		`M1_FLASH: On PCD3 extension (marked M1)`
		`M2_FLASH: On PCD3 extension (Marked M2)`
		`SL0FLASH: In I/O slot 0`
		`SL1FLASH: In I/O slot 1`
		`SL2FLASH: In I/O slot 2`
		`SL3FLASH: In I/O slot 3`

DRVName	FSCREATE FSCPRESS FSGETSIZ	The device name is passed to the function. Valid device names are identical to the one described in the FileName parameter.

GroupID	FCREATE
	DCREATE
	SWRITE
		Defines to which group the create file / directory belongs. One and only one group can be used.
		Possible groups are:
		`0x00: // File/Dir belongs to no group`
		`0x01: // Reserved (Calls return error)`
		`0x02: // File/Dir belongs to CONFIG Group`
		`0x04: // File/Dir belongs to DWNLD Group`
		`0x08: // File/Dir belongs to WEB Group`
		`0x10: // File/Dir belongs to USER1 Group`
		`0x20: // File/Dir belongs to USER2 Group`
		`0x40: // File/Dir belongs to USER3 Group`
		`0x80: // File/Dir belongs to USER4 Group`
		Specifying a 0x00 as value (was the default value of previous version) creates files or directories with NO_GROUP. This file can then be accessed independently of the given GroupAccess parameter (see below)
		Specifying 0x01 as value, the function call will return an error.

GroupAccess	FCREATE DCREATE DELETE SWRITE	This parameter allows to access files / directories belonging to one of the given groups. Any "or" combination of the above defined group can be defined. Creating a file or a directory is only possible within a directory with a group belonging to the combination of given groups. Deleting a file / directory is only possible if files / directories / sub-directories and files belonging to subdirectories with a group belonging to the combination of given groups. Specifying a 0x00 as value (was the default value of previous version) allows access to all groups.

Name	Used	Description
AccessType	FOPEN	Defines the access type when opening a file. Valid values are: `1 (B#16#1)` → Read only. An attempt to write to the file returns an error `2 (B#16#2)` -> Write only. An attempt to read to the file returns an error `3 (B#16#3)` → Read / Write. The file can be read and written.
AccessType	FSEEK	Defines the pointer(s) that will be updated when positioning in a file. Valid values are: `1 (B#16#1)` → Read only. Only the Read Pointer is updated. `2 (B#16#2)` → Write only. Only the Write Pointer is updated. `3 (B#16#3)` → Read / Write. Both pointers are updated.
SeekPos	FSEEK SREAD	Seek in current open file relative to the current position (positive or negative). A value of 0 reset the pointer(s) to the beginning of the file.

Buffer	FWRITE
	FREAD
	SWRITE
	SREAD
		The Buffer shall be built as an ***ANY*** pointer. It can point to DB, merker, input and output areas. The length is limited to 256 bytes, e.g.
		`p#DB10.DBX0.0 BYTE 20`
		will copy 20 bytes of DB 10, starting at offset 0.
		`p#M100.0 DINT 20`
		will copy 20 double words (= 80 bytes) of merker area, starting at offset 100.

WrAttr	FWRITE	Defines where the data shall be written. Valid values are: `16` → Data is written (=updated) at the current location of the Write Pointer, i.e. the data in the file is overwritten. This option is only valid on internal memory (SRAM / SDRAM) and an error is returned for FLASH file system. `17` → Data is appended at the end of the file.
Handle (out)	FCREATE FOPEN	An identifier (handle) that identifies the file. This identifier is returned after a successful FCREATE or FCLOSE and becomes invalid after a FCLOSE. If a negative value is returned, an error occurs while calling this function. The file is NOT open.
Handle (in)	FSEEK FWRITE FREAD FLENGTH FCLOSE	An identifier (handle) that identifies the file.

Name	Used	Description
RetVal	DCREATE FSEEK FWRITE FREAD FLENGTH FCLOSE DELETE SWRITE SREAD FSCREATE FSCPRESS FSGETSIZ	A return value. If smaller than 0, this means that an error occurs during the call. Refer to the error code list hereafter. When the function has been successfully performed, the return code is 0. For FREAD, the exact number of bytes transferred is returned. 0 means that the no bytes have been read. For FLENGTH, the file length is returned. 0 means that the file is empty.

2.2 Error codes

In case of success all FB's return in RetVal either 0 (zero) or a positive value. A negative value indicates an error. Below is a list of the error codes:

```
SFB_FILESYS_PLC_ERROR               -101,     0xFFFFFF9B
FS_WRONG_TYPE                       -100,     0xFFFFFF9C
FS_DEVICE_NOT_FOUND                 -99,      0xFFFFFF9D
FS_BAD_PARAMETER                    -98,      0xFFFFFF9E
FS_INVALID_ARGUMENT                 -97,      0xFFFFFF9F
FS_FILE_NOT_FOUND                   -96,      0xFFFFFFA0
FS_INVALID_FILENAME                 -95,      0xFFFFFFA1
FS_INVALID_GROUP                    -94,      0xFFFFFFA2
FS_INVALID_LEVEL                    -93,      0xFFFFFFA3
FS_INVALID_ACCTYPE                  -92,      0xFFFFFFA4
FS_INVALID_DRIVE_NAME               -91,      0xFFFFFFA5
FS_INVALID_DIRECTORY_NAME           -90,      0xFFFFFFA6
FS_FILE_ALREADY_EXIST               -89,      0xFFFFFFA7
FS_NOT_ENOUGH_SPACE                 -88,      0xFFFFFFA8
FS_TOO_MANY_OPEN_FILES              -87,      0xFFFFFFA9
FS_FILE_NOT_OPEN                    -86,      0xFFFFFFAA
FS_FILE_ALREADY_OPEN                -85,      0xFFFFFFAB
FS_INVALID_ACCESS_TYPE              -84,      0xFFFFFFAC
FS_INVALID_FILE_TYPE                -83,      0xFFFFFFAD
FS_INVALID_WRITE_ATTR               -82,      0xFFFFFFAE
FS_INVALID_BUFFER                   -81,      0xFFFFFFAF
FS_WRITE_ERROR                      -80,      0xFFFFFFB0
FS_READ_ERROR                       -79,      0xFFFFFFB1
FS_DAS_ACCESS_REFUSED               -78,      0xFFFFFFB2
FS_ACCESS_DENIED                    -77,      0xFFFFFFB3
FS_INV_FILE_DESCR                   -76,      0xFFFFFFB4
FS_INVALID_USER                     -75,      0xFFFFFFB5
FS_INVALID_REGFLAGS                 -74,      0xFFFFFFB6
FS_REG_ENTRY_TABLE_FULL             -73,      0xFFFFFFB7
FS_INVALID_REGID                    -72,      0xFFFFFFB8
FS_FILE_SYSTEM_CHECK_ERROR          -71,      0xFFFFFFB9
FS_INV_ENV_NAME                     -70,      0xFFFFFFBA
FS_ENV_NOT_LOADED                   -69,      0xFFFFFFBB
FS_ENV_NAME_ALREADY_EXIST           -68,      0xFFFFFFBC
FS_INVALID_OPERATION                -67,      0xFFFFFFBD
FS_INVALID_FLASH_VALUE              -66,      0xFFFFFFBE
FS_FAILED_FLASH_OPERATION           -65,      0xFFFFFFBF
FS_COMPRESSION_ERROR                -64,      0xFFFFFFC0
FS_DEVICE_BUSY                      -63,      0xFFFFFFC1
FS_OPERATION_RESCHEDULED            -62,      0xFFFFFFC2
FS_NOT_IMPLEMENTED                  -61,      0xFFFFFFC3
FS_INTERNAL_ERROR                   -60,      0xFFFFFFC4
```

Remark: Special take shall be taken when the `FS_DEVICE_BUSY` code is returned. This means that the current device is currently performing an action (e.g. recovering freed blocks) which takes too much time. The function call can NOT wait until this action is finished. The user shall retry later in order to perform its action.

Remark: When the returned error code is `FS_NOT_ENOUGH_SPACE`, either the FLASH is really full and nothing else can be written on it or the FLASH is not full but no free blocks are available. In that case, it may be necessary to call the FSCPRESS (FB462) to recover these blocks again.

3 FB Description

3.1 FB “FCREATE” [FB 450]

This FB creates a file as defined in the `FileName` argument.

```
CALL  FB 450 , DB450
       FileName   := [STRING[64]] See FileName in § 2.1 above
       GroupID    := [BYTE]       See GroupID in § 2.1 above
       GroupAccess:= [BYTE]       See GroupAccess in § 2.1 above
       Handle     := [DINT]       See Handle (out) in § 2.1 above
```

The FB 450 (“FCREATE”) will call the SFB 450. The FB call requires a (multi-)instance data block, referenced here has DB450.

3.2 FB “DCREATE” [FB 451]

This FB creates a directory as defined in the `FileName` argument.

```
CALL  FB451 , DB451
       FileName   := [STRING[64]] See FileName in § 2.1 above
       GroupID    := [BYTE]       See GroupID in § 2.1 above
       GroupAccess:= [BYTE]       See GroupAccess in § 2.1 above
       RetVal     := [DINT]       See RetVal in § 2.1 above
```

The FB 451 (“DCREATE”) will call the SFB 451. The FB call requires an (multi-)instance data block, referenced here has DB451.

3.3 FB “FOPEN” [FB 452]

This FB opens a file as defined in the `FileName` argument, with the given access type.

```
CALL  FB452 , DB452
       FileName   := [STRING[64]] See FileName in § 2.1 above
       AccessType := [BYTE]       See AccessType in § 2.1 above
       Handle     := [DINT]       See Handle (out) in § 2.1 above
```

The FB 452 (“FOPEN”) will call the SFB 452. The FB call requires an (multi-)instance data block, referenced here has DB452.

3.4 FB “FSEEK” [FB 453]

This FB seeks into a file previously open with FB 452 or created with FB 450. Refer to 1.2 above for information about the seek algorithm.

```
CALL  FB453 , DB453
       Handle     := [DINT]  See Handle (in) in § 2.1 above
       SeekPos    := [DINT]  See SeekPos in § 2.1 above
       AccessType := [BYTE]  See AccessType in § 2.1 above
       RetVal     := [DINT]  See RetVal in § 2.1 above
```

The FB 453 (“FSEEK”) will call the SFB 453. The FB call requires an (multi-)instance data block, referenced here has DB453.

3.5 FB “FWRITE” [FB 454]

This FB write data into a file previously open with FB 452 or created with FB 450.

```
CALL  FB454 , DB454
       Handle     := [DINT]  See Handle (in) in § 2.1 above
       WrAttr     := [BYTE]  See WrAttr in § 2.1 above
       Buffer     := [ANY]   See Buffer in § 2.1 above
       RetVal     := [DINT]  See RetVal in § 2.1 above
```

The FB 454 (“FWRITE”) will call the SFB 454. The FB call requires an (multi-)instance data block, referenced here has DB454.

3.6 FB “FREAD” [FB 455]

This FB reads data from a file previously open with FB 452 or created with FB 450.

```
CALL  FB455 , DB455
       Handle     := [DINT]  See Handle (in) in § 2.1 above
       Buffer     := [ANY]   See Buffer in § 2.1 above
       RetVal     := [DINT]  See RetVal in § 2.1 above
```

The FB 455 (“FREAD”) will call the SFB 455. The FB call requires an (multi-)instance data block, referenced here has DB455.

3.7 FB “FLENGTH” [FB 456]

This FB returns the length of a file previously open with FB 452 or created with FB 450.

```
CALL  FB456 , DB456
       Handle     := [DINT]  See Handle (in) in § 2.1 above
       RetVal     := [DINT]  See RetVal in § 2.1 above
```

The FB 456 (“FLENGTH”) will call the SFB 456. The FB call requires an (multi-)instance data block, referenced here has DB456.

3.8 FB “FCLOSE” [FB 457]

This FB closes a file previously open with FB 452 or created with FB 450.

```
CALL  FB457 , DB457
       Handle     := [DINT]  See Handle (in) in § 2.1 above
       RetVal     := [DINT]  See RetVal in § 2.1 above
```

The FB 457 (“FCLOSE”) will call the SFB 457. The FB call requires an (multi-)instance data block, referenced here has DB457.

3.9 FB “DELETE” [FB 458]

This FB deletes a file (if a filename is given as `FileName` argument) or deletes a directory (if a directory name is given as `FileName` argument) together with all its content.

```
CALL  FB458 , DB458
       FileName   := [STRING[64]]See FileName in § 2.1 above
       GroupAccess:= [BYTE]      See GroupAccess in § 2.1 above
       RetVal     := [DINT]      See RetVal in § 2.1 above
```

The FB 458 (“Delete”) will call the SFB 458. The FB call requires an (multi-)instance data block, referenced here has DB458.

3.10 FB “SWRITE” [FB 459]

This FB performs the open (or create if file does not yet exist), the writing (at the end of the file) and the close in one single call.

```
CALL  FB459 , DB459
       FileName    := [STRING[64]]See FileName in § 2.1 above
       GroupID     := [BYTE]      See GroupID in § 2.1 above
       GroupAccess:= [BYTE]       See GroupAccess in § 2.1 above
       Buffer      := [ANY]       See Buffer in § 2.1 above
       RetVal      := [DINT]      See RetVal in § 2.1 above
```

The FB 459 (“SWRITE”) will call the SFB 459. The FB call requires an (multi-)instance data block, referenced here has DB459.

3.11 FB “SREAD” [FB 460]

This FB performs the open (or create if file does not yet exist), the seek operation at the required file position, the read and the close in one single call.

```
CALL  FB460 , DB460
       FileName    := [STRING[64]]See FileName in § 2.1 above
       Buffer      := [ANY]       See Buffer in § 2.1 above
       Offset      := [DINT]      See SeekPos in § 2.1 above
       RetVal      := [DINT]      See RetVal in § 2.1 above
```

The FB 460 (“SREAD”) will call the SFB 460. The FB call requires an (multi-)instance data block, referenced here has DB460.

For this call, the seek position shall be bigger or equal to 0 as the file is open just before the seek operation. Opening a file sets the read and write pointer at the beginning of the file.

3.12 FB “FSCREATE” [FB 461]

This FB performs the formatting / creation of a file system on a given device. This operation is asynchronous; more than one call is required until the operation is finished.

The real operation is started when the `Req` parameter is 1. During the first call, the `busy` parameter is set to 1, the `done` parameter is set to 0 and the `retval` parameter is set to `FIRST_CALL` (0x7001). The user program shall call this function until the `busy` parameter is set to 0. All calls returning with the `busy` parameter set to 1 will set the `retval` to `INTERIM_CALL` (0x7002).

When the `busy` parameter is set to 0, the `retval` parameter can be evaluated. The function has successfully completed when a 0 is returned. Any further call with the `req` parameter set to 1 will launch again the formatting / creation of the file system. The `retval` parameter can have a negative value (refer to §2.2) meaning that the action could not performed successfully. In that case, the `done` parameter is kept to 0. Any further call to the function will return the same error code until the function is called with the `req` parameter set to 0. When set again to 1, the creation will start again.

As long as the `busy` parameter is set to 1, the `req` parameter is not evaluated.

If the function is not yet active and the function is called with the `req` parameter set to 0, the `retval` parameter is set to `CALL_WITHOUT_EXEC` (0x7000).

The `force` parameter can be set to 0 or 1. Setting a 0, allows calling the function without forcing the formatting of the device if it is already present. Setting a 1, allows calling the function and forcing the formatting of the device even if a file system is already present on the device. This operation will delete all data on the device and perform a formatting. If the `force` parameter is 0

and the device is present but not recognized as a file system device, the formatting will take place. All data previously stored on that device will be lost, e.g. if backup data were written on that device.

The `DRVName` parameter is used to determine which device shall be formatted. Refer to DRVName in § 2.1 above for available drive names.

The user has the possibility to reformat the given device with a new block size. For PCD7.R5xx devices, the `BlockSize` can be 512, 1024 (1 KB), 2048 (2KB), 4096 (4KB) and 8192 (8KB). The default `BlockSize`, is set to 2048. The given `BlockSize` is compared with the current value used for the device. If the block sizes are different and the `BlockSize` parameter is valid, the device is reformatted, independently if the `force` parameter is set or not. If the block sizes are identical, then the `force` parameter is evaluated as described in the previous paragraph.

The `BlkNbr` and `MNbr` parameters are currently not used and shall be set respectively to, 256 (`BlkNbr`) and 32 (`MNbr`).

The `RetVal` parameter is either set to `CALL_WITHOUT_EXEC` (0x7000), `FIRST_CALL` (0x7001), `INTERIM_CALL` (0x7002) during the execution of the function. The real function return value is set when the `Busy` parameter goes from 1 to 0. A 0 value returned at that moment means that everything has been successfully performed and the device can be accessed with the other file system related FB's.

The `Busy` parameter is set to 1 as long as the function is under execution and is reset to 0 when the function has been finished.

The `Done` parameter is set to 0 as long as the function is under execution. It is set to 1 if the function has been successfully executed and remains to 0 in case of error.

The FB 461 call can be called as defined hereunder:

```
CALL  FB 461 , DB461
       Req      := [BOOL]    See above for description
       Force    := [BOOL]    See above for description
       FSName   := [STRING[16]]  See DRVName in § 2.1 above
       BlockSize:= [WORD]    See above for description
       BlkNbr   := [WORD] W#16#100    See above for description
       MNbr     := [WORD] W#16#20     See above for description
       RetVal   := [DINT]         See RetVal in § 2.1 above
       Busy     := [BOOL]    See above for description
       Done     := [BOOL]    See above for description
```

The FB 461 ("FSCREATE") will call the SFB 461. The FB call requires an (multi-)instance data block, referenced here has DB461.

Important remark: Calling this function with the `force` parameter set to 1 or with a block size parameter different from the one used on the device will delete all data on the device.

3.13 FB "FSCPRESS" [FB 462]

This FB performs the compression / recovery for freed blocks of a file system on a given device. This operation is asynchronous; more than one call is required until the operation is finished.

On the PCD7.R55x FLASH modules, a block is considered either as free (not used yet), as busy (currently used) and freed (has been used at one moment). Freed blocks can not been used until the sector containing the block is erased (all bits set to 1). Only at that moment, a freed block can be re-entered in the free list of blocks.

Internally to the file system, some blocks may be marked as freed, but mainly a block is marked as freed when a file / directory is deleted, all associated blocks being released.

Internally to the file system, the compression (recovery of freed block) is automatically launched when some criteria are met, e.g. the number of freed blocks is 80% of total number of blocks or

when the number of freed blocks is bigger than the number of free blocks if this number is less that 1/4th of the total number of blocks. However, this operation can occur at any time and during this operation, the file system is marked as busy. All calls to the file system FB's will then return the `FS_DEVICE_BUSY` code.

This FB can be used by the user to force the compression of the device, even if the previous criteria are not met, e.g. if a file is deleted, the user may want to immediately recover all blocks related to that file.

The real operation is started when the `Req` parameter is 1. During the first call, the `busy` parameter is set to 1, the `done` parameter is set to 0 and the `retval` parameter is set to `FIRST_CALL` (0x7001). The user program shall call this function until the `busy` parameter is set to 0. All calls returning with the `busy` parameter set to 1 will set the `retval` to `INTERIM_CALL` (0x7002).

When the `busy` parameter is set to 0, the `retval` parameter can be evaluated. The function has successfully completed when a 0 is returned. Any further call with the `req` parameter set to 1 will launch again the compression of the given device. The `retval` parameter can have a negative value (refer to §2.2) meaning that the action could not be performed successfully. In that case, the `done` parameter is kept to 0. Any further call to the function will return the same error code until the function is called with the `req` parameter set to 0. When set again to 1, the creation will start again.

As long as the `busy` parameter is set to 1, the `req` parameter is not evaluated.

If the function is not yet active and the function is called with the `req` parameter set to 0, the `retval` parameter is set to `CALL_WITHOUT_EXEC` (0x7000).

The `DRVName` parameter is used to determine which device shall be compressed. Refer to DRVName in § 2.1 above for available drive names.

The `Busy` parameter is set to 1 as long as the function is under execution and is reset to 0 when the function has been finished.

The `Done` parameter is set to 0 as long as the function is under execution. It is set to 1 if the function has been successfully executed and remains to 0 in case of error.

The `RetVal` parameter is either set to `CALL_WITHOUT_EXEC` (0x7000), `FIRST_CALL` (0x7001), `INTERIM_CALL` (0x7002) during the execution of the function. The real function return value is set when the `Busy` parameter goes from 1 to 0.

The FB 462 call can be called as defined hereunder:

```
CALL  FB462 , DB462
 Req    := [BOOL]      See above for description
 DRVName:= [STRING[16]]    See DRVName in § 2.1 above
 Busy   := [BOOL]      See above for description
 Done   := [BOOL]      See above for description
 RetVal := [DINT]      See RetVal in § 2.1 above
```

The FB 462 ("FSCPRESS") will call the SFB 462. The FB call requires an (multi-)instance data block, referenced here has DB462.

Important remark: Because of the used technology on the PCD7.R55x devices, this function requires time to execute. The time required until the `busy` parameter is back to 0 can be between 1 second (no blocks had to be recovered) and 4 minutes (at least 2 blocks in each sectors of the device could be recovered). The power OFF of the device is strictly forbidden. In case of power loss during that time, the compress algorithm is restarted when the device is powered ON again and time is required until the device can be accessed (read/write) from the user program.

3.14 FB “FSGETSIZ” [FB 463]

This FB gets some information concerning the given device. The returned information is:

- The file system as it was created.
- The currently used size (including freed blocks) (see 3.13 above)
- The currently free size

```
CALL  FB463 , DB463
       DRVName := [STRING[16]]   See DRVName in § 2.1 above
       TSIZE   := [DINT]         Total size of device
       FSIZE   := [DINT]         Free size
       USIZE   := [DINT]         Used size
       RetVal  := [DINT]         See RetVal in § 2.1 above
```

The FB 463 (“FSGETSIZ”) will call the SFB 463. The FB call requires an (multi-)instance data block, referenced here has DB463.

3.15 FB “FSDEVINF” [FB 464]

This FB gets some information concerning the given device. The returned information is:

- The file system size as it was created;
- The current block size;
- The number of blocks allocated for the file system;
- The calculated maximum number of simultaneous open files;
- The number of currently open files.

```
CALL  FB464 , DB464
     DRVName             := [STRING[16]] See DRVName in § 2.1 above
     DEVSIZE             := [DINT] Total size of device
     DEVBLOCKSIZE        := [DINT] BlockSize
     DEVBLOCKNBR         := [DINT] Nbr of blocks
     DEVMNBROFOPENFILES := [DINT] Maximum number of simultaneously open files
     NBROFOPENFILES      := [DINT] Number of currently open files
     RetVal              := [DINT] See RetVal in § 2.1 above
```

The FB 464 (“FSDEVINF”) will call the SFB 464. The FB call requires an (multi-)instance data block, referenced here has DB464.

3.16 FB “FSDEVSTA” [FB 465]

This FB gets the current status of the given device. The returned information is:

- Device Status. The returned status is either
  - `FS_NO_ERROR [0]`: File System is OK
  - `FS_DEVICE_BUSY [-63]`: File System is currently not accessible
  - `FS_FILE_SYSTEM_CHECK_ERROR [-71]`: Device was found, but the file system could not be re-created.
  - `FS_DEVICE_NOT_FOUND [-99]`: No such device present.

```
CALL  FB465 , DB465
     DRVName := [STRING[16]] See DRVName in § 2.1 above
     STATE   := [DINT]       See above
```

The FB 465 (“FSDEVSTA”) will call the SFB 465. The FB call requires an (multi-)instance data block, referenced here has DB465.