The Frame Distribution (Fd)
User's guide

A. Masserot, B. Mours, D. Verkindt
(LAPP-CNRS Annecy)

Version: v8r15
January 16rd, 2016

TABLE OF CONTENTS

1. INTRODUCTION 5

2. BASIC USE OF THE FD LIBRARY 6

2.1. INTRODUCTION 6
2.2. DESCRIPTION OF THE BASIC FDIO FUNCTIONS 6
2.2.1. FdIONew 6
2.2.2. FdIOParseAndIni 6
2.2.3. FdIOGetFrame 6
2.2.4. FdIOPutFrame 7
2.2.5. FdUpdateState 7
2.2.6. FdStatBuild 7
2.3. EXAMPLE OF APPLICATION: FDMONI 7
2.4. INFORMATION FOR ADVANCED DEVELOPERS TO SEND/RECEIVE FRAMES OVER CM 7
2.4.1. Function to put a frame in a Cm messags: FdPutFrameInMessage 7
2.4.2. Function to get a frame from a Cm message: FdGetFrameFromMessage 8
2.4.3. Function to request frames via Cm: FdRequestFrames 8
2.4.4. Function to remove a frame request: FdRemoveFramesRequest 8
2.4.5. Function to request the list of channel: FdRequestListOfChannels 8

3. BASIC PROGRAMS AND TOOLS 9

3.1. FDIOSERVER 9
3.2. EXAMPLES OF FDIOSERVER CONFIGURATIONS 9
3.2.1. simChannels.cfg 9
3.2.2. copyFile.cfg 9
3.2.3. dupChannels.cfg 9
3.2.4. trendFrameBuilder.cfg 10
3.2.5. demodulation.cfg 10
3.2.6. writeDir.cfg + readDirDy.cfg 10
3.2.7. cmSend.cfg + cmReceive1.cfg, cmReceive2.cfg and cmReceive3.cfg 10
3.2.8. frameMerger.cfg + toFrameMerger_1.cfg and toFrameMerger_2.cfg 10
3.2.9. playback.cfg 10
3.2.10. softwareInjections.cfg 10
3.2.11. noConfigCheck.cfg: 11
3.3. FDGETCHANNELSLIST 11
3.4. FDSEND 11
3.5. FDSTAT 11
3.6. FDWRITE 11

4. CONFIGURATION PARAMETERS 12

4.1. CONFIGURATION ACCESS 12
4.2. KEYWORDS TO CONTROL GLOBAL PARAMETERS 12
4.2.1. FD_CH_PREFIX 12
4.2.2. FD_DEBUGLVL 12
4.2.3. FD_NO_CONFIG_CHECK 12
4.2.4. FDIN_NO_FR_MSG_TIMEOUT 12
4.2.5. FDIN_CM_TIMEOUT 12
4.2.6. FDOUT_CM_TIMEOUT 12
4.3. KEYWORDS TO DEFINE AND CONTROL THE SOURCE OF INPUT FRAMES 12
4.3.1. FDIN_CM 13
4.3.2. FDIN_CM_CONNECT 13
4.3.3. FDIN_DATA_SENDER 13
4.3.4. FDIN_DIR 13
4.3.5. FDIN_FILE 13
4.3.6. FDIN_FRAME_MERGE 14
4.3.7. FDIN_MAX_SPEED 14
4.3.8. FDIN_MIN_LATENCY 14

4.3.9. FDIN_NEW_FRAME *15*
4.3.10. FDIN_PLAYBACK *15*
4.3.11. FDIN_SLEEP *15*
4.3.12. FDIN_TAG *15*
4.4. KEYWORDS TO CONTROL THE INPUT FRAMES CONTENT 16
4.4.1. FDIN_ADD_SURROGATE_CHANNEL *16*
4.4.2. FDIN_ADD_SURROGATE_DQ_CHANNEL *16*
4.4.3. FDIN_ADD_TEST_CHANNEL *16*
4.4.4. FDIN_BUT_FILTER *17*
4.4.5. FDIN_COMBINE_CHANNELS *17*
4.4.6. FDIN_COMPRESSION *17*
4.4.7. FDIN_DUPLICATED_CHANNELS_CHECK *18*
4.4.8. FDIN_DUPLICATED_CHANNELS_REMOVE *18*
4.4.9. FDIN_FFT_FILTER *18*
4.4.10. FDIN_FRAME_DURATION *18*
4.4.11. FDIN_NOMSG *18*
4.4.12. FDIN_RANGE_GATING *19*
4.4.13. FDIN_RESCALE_CHANNEL *20*
4.4.14. FDIN_STAT *20*
4.4.15. FDIN_RECAST_CHANNELS *20*
4.4.16. FDIN_REJECT_EVENTS *21*
4.4.17. FDIN_RENAME_CHANNEL *21*
4.4.18. FDIN_SELECT_CHANNELS *21*
4.4.19. FDIN_S_INJECTION_FILE *21*
4.4.20. FDIN_S_INJECTION_EVT_NAME *21*
4.4.21. FDIN_S_INJECTION_RESCHEDULE *22*
4.4.22. FDIN_S_INJECTION_RESCALE *22*
4.4.23. FDIN_S_INJECTION_CHANNEL *22*
4.4.24. FDOUT_SET_PROC_TYPE *22*
4.4.25. FDIN_FSHIFT *22*
4.4.26. FDIN_DEMOD *23*
4.4.27. FDIN_GET_PHASE *23*
4.4.28. FDIN_GET_MODULUS *23*
4.5. KEYWORDS TO CONTROL THE FRAME OUTPUT(S) 24
4.5.1. FDOUT_BUT_FILTER *24*
4.5.2. FDOUT_DUPLICATED_CHANNELS_CHECK *24*
4.5.3. FDOUT_DUPLICATED_CHANNELS_REMOVE *24*
4.5.4. FDOUT_CLEAN_DIR *24*
4.5.5. FDOUT_CLEAN_DIR_FFL *24*
4.5.6. FDOUT_CM *24*
4.5.7. FDOUT_CM_SERVER *25*
4.5.8. FDOUT_COMBINE_CHANNELS *25*
4.5.9. FDOUT_COMPRESSION *25*
4.5.10. FDOUT_CONSUMER *26*
4.5.11. FDOUT_CONVERT_SERDATA *26*
4.5.12. FDOUT_DELAY *26*
4.5.13. FDOUT_FFT_FILTER *26*
4.5.14. FDOUT_FILE *26*
4.5.15. FDOUT_FREQUENCY_CUT *26*
4.5.16. FDOUT_FRAME_DURATION *26*
4.5.17. FDOUT_RECAST_CHANNELS *26*
4.5.18. FDOUT_RENAME_CHANNEL *26*
4.5.19. FDOUT_REJECT_EVENTS *27*
4.5.20. FDOUT_RESCALE_CHANNEL *27*
4.5.21. FDOUT_SELECT_CHANNELS *27*
4.5.22. FDOUT_SET_NEXT_STOP *27*
4.5.23. FDOUT_SET_PREFIX *27*
4.5.24. FDOUT_SET_PROC_TYPE *27*
4.5.25. FDOUT_STAT *27*
4.5.26. FDOUT_TREND_FRAME *27*

5. CM MESSAGES AVAILABLE 28

5.1. CM CONTROL MESSAGES WHICH COULD BE SENT TO A FD PROCESS 28
5.1.1. Message FdGetChannelsList (formally FdFrameChannels) 28
5.1.2. Message FdAddCmOuput (formally FdAddFrame) 28
5.1.3. Message FdRemoveCmOutput (formally FdRemoveFrame) 28
5.1.4. Message FdAddFrameMergerSource 29
5.1.5. Message FdRemoveFrameMergerSource 29
5.1.6. Message FdSetNextStop 29
5.2. CM MESSAGES RETURNED BY A FD PROCESS 29
5.2.1. Message FdChannelsList 29
5.2.2. Message FdFrame 30

6. PACKAGE INSTALLATION 31

6.1.1. Virgo Software Package organization 31
6.1.2. Installing CMT 31
6.1.3. Installing PackageManagement 31
6.1.4. Installing the "Fd" package 32
6.1.5. Configure and testing Fd/Cm 32
6.1.6. Summary of the commands which are useful to keep in your .cshrc 32
6.1.7. Installing the code without Pm or debugging cmt make problems 32

7. HISTORY 33

7.1. V8R00 (OCTOBER 2ND, 2014) 33
7.2. V8R00P1 (OCTOBER 8TH, 2014) 33
7.3. V8R01 (OCTOBER 18TH, 2014) 33
7.4. V8R02 (NOVEMBER 6TH, 2014) 33
7.5. V8R03 (NOVEMBER 18TH, 2014) 33
7.6. V8R04 (DECEMBER 7TH, 2014) 33
7.7. V8R04P1 (DECEMBER 13TH, 2014) 34
7.8. V8R05 (JANUARY 7TH, 2015) 34
7.9. V8R06 (JANUARY 27TH, 2015) 34
7.10. V8R07 (FEBRUARY 28TH, 2015) 34
7.11. V8R08 (MARCH 5TH, 2015) 34
7.12. V8R09 (APRIL 12TH, 2015) 34
7.13. V8R10 (APRIL 27TH, 2015) 35
7.14. V8R11 (MAY 17TH, 2015) 35
7.15. V8R12 (MAY 26TH, 2015) 35
7.16. V8R13 (AUGUST 18TH, 2015) 35
7.17. V8R14 (JANUARY 3RD, 2016) 35
7.18. V8R15 (JANUARY 16, 2016) 35

1. Introduction

This document is the user's guide for the Fd library and its associated applications like the FdIOServer.

The Fd software has evolved from its early use in the Virgo DAQ system of transporting frames over the network using the Cm package to a more general I/O framework for online and offline applications. Its main purpose is to provide data in the frame format from and to files, shared memory and network connection. Basic data conditioning tools like channel filtering, resampling are now included in Fd as well as frame manipulation like channels selection, frame resizing, trend frame building and frame merging between different streams. The frames handled by the Fd package could contain any frame structures, from time series to events.

A user can build its own application with very few functions described in section 2. The main ones are the FdIOGetFrame and FdIOPutFrame functions to access the data. The Fd logic as also evolved over time. Since version 8, the sequence of operations performed by FdIOGetFrame and FdIOPutFrame is fully determined by the configuration files, especially by the order of the requested actions which will be performed as listed. This allows the construction of complex pipelines with multiple outputs for instance. The configuration parameters are described in section 4.

The FdIOServer, described in section **Error! Reference source not found.**, is the main application of the Fd package. This application is a simple loop to get and put frames according to a configuration file. This is the basic building block of a DAQ system, once the data are in the frame format, since it provides access to all Fd tools.

Frames could be exchanges between process by writing frame files in a directory located on disk (for exchange between different machines) or in memory (/dev/shm for exchanges on the same machine). When sending frames from on computer to another one over the network, the frames are written in a memory block and put in a Cm "message": the basic chunk of data exchange over TCP/IP used by the Cm library.

When exchanging data between processes via /dev/shm memory (on the same machine) or network (usually on different machine) the data are always stored in frames which are formatted in a memory like a single frame would be written on disk. Therefore, in the case of network exchange, the Frame Library provides the needed conversion (little/big endian, 32/64 bits) when using a heterogeneous set of machines.

Some of the configuration parameters could by changed at run time via Cm messages described in section 5.

The Fd library is build on top of the Frame library (Fr packages), the Cfg package for configuration, log files, messages and error handling and the Cm package for data exchange over network. The package installation is described in section 6.

2. Basic use of the Fd library

2.1. Introduction

An application which needs to read and/or writes frames need to call very few functions: FdIONew, FdIOParseAndIni, FdIOGetFrame, FdIOPutFrame. The following C code illustrates their use.

```
#include "Fd.h"

int main(int  argc, char *argv[])
{
  FrameH *frame;
  FdIO *fdIO

  fdIO = FdIONew(argc, argv); /*-----------------------create the basic Fd object-*/
  if(fdIO == NULL) exit(EXIT_FAILURE);

  /*------------------enrich here the Cfg parser (fdIO->parse) with your own keys-*/

  FdIOParseAndIni(fdIO); /*-parse the config. to extract Fd(and users) parameters-*/

  while (!CfgFinished()) {  /*--- run as long data arrive and do not ask for stop-*/

    frame = FdIOGetFrame(fdIO); /*---------------------------------extract a frame-*/
    if(frame == NULL) continue; /*no frame arrive during the FdIOGetFrame timeout-*/

    /*-----------------------------------------------do your frame processing here-*/

    FdIOPutFrame(fdIO, frame);} /*--output frame and free the corresponding space-*/

  return(0);
}
```

2.2. Description of the basic FdIO functions

2.2.1. FdIONew

```
FdIO* FdIONew(int  argc, char *argv[]);
```

This function creates a FdIO structure, with its associated parser and initializes the Cfg library. Following the Cfg convention, the configuration file name which should be in argv[1] (first parameter of the process).
Comment for advanced developers:

- This function creates a parser object which includes the Cfg keys. This parser (fdIO->parser) could be enriched by the user to add its own keys.
- This function was named FdIOIni before version 8 of Fd. It has been renamed to avoid confusion; although the FdIOIni is still temporarily available.
- The FdIONew function starts by a call to CfgIdle. If a developer wants to make directly the call to CfgIdle, he can use the FdIONewNoCfgIdle function.

2.2.2. FdIOParseAndIni

```
void  FdIOParseAndIni (FdIO* fdIO);
```

This function parses the configuration parameters specific to FdIO and do the needed Fd initialization after parsing.
If the parser has been enriched by the key specific to the user application, it can tell if all keywords have been properly handled or if there are orphan configuration part not used by the parser, which could be an indication of error.
This function was named FdIOParse before version 8 of Fd. It has been renamed to avoid confusion; nevertheless the old FdIOParse function is still temporarily available.

2.2.3. FdIOGetFrame

```
FrameH* FdIOGetFrame (FdIO* fdIO);
```

This function gets a frame from the "input" (file/network/loca generation) defined by the configuration of the process. Some additional processing could be performed on the input frames, according to the configuration of the process (change of the frame duration, channels selection, channels filtering).

2.2.4. FdIOPutFrame

```
void FdIOPutFrame (FdIO* fdIO, FrameH* frame);
```

This function outputs a frame according to the process configuration (or the default parameter defined by FdIONew).
Some additional processing could be performed when output the frame, according to the configuration of the process (change of the frame duration, channels selection, channels filtering).

2.2.5. FdUpdateState

By default a call to FdIOGetFrame and FdIOPutFrame update de program state (CfgServerActive, CfgServerGolden, or CfgServerError), accordingly to the possible problem of the input/output frames.

If the user wants to keep the control of the state update, he must use the function:

*int FdUpdateState(FdIO *fdIO, int state)*

- A single call with a negative state value will disable these updates.
- A call with the state value equal to zero will return the worse state value (CfgServerActive, CfgServerGolden, or CfgServerError) of the FdIO objects, without updating the program state.
- A call with a state value equal to CfgServerActive, CfgServerGolden, or CfgServerError, will take the worse value of this input value and all the FdIO actions and will update the program state accordingly.

2.2.6. FdStatBuild

This function:

- Returns a string which could be used as a "user info" message for the CfgMsrAddUserInfo function.
- Fill the fdIO->serData string which could be used to record various parameters of the FdIO actions.

This function performs the same actions as the FDIN_STAT or FDOUT_STAT keys, except the CfgMsrAddUserInfo function is not called and the FrSerData is not added to the frame. This function could be useful for a dedicated application which would like to build its own user info and FrSerData, enriching the FdIO one.

Remark for developers: each FdAction has two strings: action->userInfo and action->serData
It is possible to disable the Cm message sends by setting the timeout value to a negative value fdIO->…

2.3. Example of application: FdMoni

The file FdMoni.c in the src directory is an example of the functions that a user may add to build its own processing using Fd.

2.4. Information for advanced developers to send/receive frames over Cm

As explained in the previous sections, a simple program needs only to call the 4 basic functions: FdIONew, FdIOParseAndIni, FdIOGetFrame, FdIOPutFrame.

In this section we present a list of other lower level functions that might be useful for an advanced developer. All these functions are group in the FdUtil.c file which just requires the Cm and Fr library. It could be use without the Cfg framework. The FdUtil.h include is enough to use them. Their use are illustrated by some of the tools or example files.

2.4.1. Function to put a frame in a Cm messags: FdPutFrameInMessage

CmMessage FdPutFrameInMessage(FrameH *frame, int compress, int *nBytes)*

This function puts the frame in a Cm message. The Cm message is created and returns by this function. Then the Cm message can be sent using the function: *CmMessageSend(message);*. The user must take care of deleting the massage after its use.

The argument *compress* is the compression type (see FrameLib). If the compress value is -1 (the recommended value), the stored frame is not touched.
The function put in *nBytes*, the number of bytes used by the frame buffer or zero in case of error. In this case, it returns also NULL.

See the FdSend.c tool as an example of use of this function.

2.4.2. Function to get a frame from a Cm message: FdGetFrameFromMessage

FrameH FdGetFrameFromMessage (CmMessage message, int compress, int *nBytes)*

This function extracts the frame from the incoming Cm message, and gives also the message frame size (in bytes).

See the FdWrite.c tool as an example of use of this function.

2.4.3. Function to request frames via Cm: FdRequestFrames

*int FdRequestFrames (char *source, char* source, char *tag, int queueSize, int nRetry);*

This function requests (sending a Cm message) to a Cm frame source *source* to send frame with the channel selection defined by *tag* to the destination *dest*. The size of the output message queue is defined by *queueSize*.

See the FdWrite.c tool as an example of use of this function.

2.4.4. Function to remove a frame request: FdRemoveFramesRequest

int FdRemoveFramesRequest (char source, char* dest);*

This function ask to Cm frame source *source* to not send anymore frame to the destination *dest*.

See the FdWrite.c tool as an example of use of this function.

2.4.5. Function to request the list of channel: FdRequestListOfChannels

*int FdSendFrameChannelsRequest (char *source)*

This function asks to the Cm frame source to receive the list of frame for a given GPS time. The GPS time is used only for the sources which can access the frame at different time like a data sender.

See the FdGetChannelsList.c tool as an example of use.

3. Basic programs and tools

The following applications are built as part of the Fd package. They provide access to basically all the features of the Fd library trough their configuration or parameters. Besides their direct use, they illustrate the possible applications of the Fd library.

3.1. FdIOServer

This is the main program to read frames from one input, process them and output them according to its configuration file. See section 4 for the description of all possible actions and keywords definition.
Use is

```
FdIOServer  configuration.cfg
```

3.2. Examples of FdIOServer configurations

Several example of configuration are given in the test directory. They are useful examples to check the Fd library. They are listed according the suggested sequence of test

3.2.1. simChannels.cfg

To test the fake frame production, channels production and recasting, and frame writing.
This is to test the following keys:

- FDIN_NEW_FRAME
- FDIN_MAX_SPEED
- FDIN_ADD_TEST_CHANNEL
- FDIN_BUT_FILTER
- FDIN_FFT_FILTER
- FDIN_RECAST_CHANNELS
- FDIN_RENAME_CHANNEL
- FDIN_RESCALE_CHANNEL
- FDIN_SET_PROC_TYPE
- FDOUT_BUT_FILTER
- FDOUT_FFT_FILTER
- FDOUT_RECAST_CHANNELS
- FDOUT_RENAME_CHANNELS
- FDOUT_RESCALE_CHANNEL
- FDOUT_STAT
- FDOUT_SET_PREFIX
- FDOUT_SET_PROC_TYPE
- FDOUT_COMPRESSION
- FDOUT_FILE

3.2.2. copyFile.cfg

To copy a file changing the frame length. You need to run first the simChannel.cfg configuration to create the test file.
This is to test the following keys:

- FDIN_FILE
- FDIN_FRAME_DURATION
- FDOUT_FRAME_DURATION
- FDIN_NOMSG
- FDOUT_FILE

3.2.3. dupChannels.cfg

To test the duplicated channel monitoring and removal as well as the frame read. It uses a specific test input file: ../test/testDupChnls.gwf.
This is to test the following keys:

- FDIN_FILE
- FDIN_STAT
- FDIN_DUPLICATED_CHANNELS_CHECK
- FDIN_DUPLICATED_CHANNELS_REMOVE
- FDOUT_DUPLICATED_CHANNELS_CHECK

- FDOUT_DUPLICATED_CHANNELS_REMOVE

3.2.4. trendFrameBuilder.cfg

To test the trend frame production, writing in multiple directory and produces an ffl.
This is to test the following keys:

- FD_CH_PREFIX
- FDOUT_CLEAN_DIR_FFL
- FDOUT_TREND_FRAME

3.2.5. demodulation.cfg

To test demodulation functions.
This is to test the following keys:

- FDIN_FSHIFT
- FDIN_DEMOD
- FDIN_GET_PHASE
- FDIN_GET_MODULUS

3.2.6. writeDir.cfg + readDirDy.cfg

To tests the frame exchange via /dev/shm, the reading process act as a frame server over Cm for dataDisplay for instance.
This is to test the following keys:

- FD_DEBUGLVL
- FDIN_DIR
- FDIN_TAG
- FDOUT_CM_SERVER

3.2.7. cmSend.cfg + cmReceive1.cfg, cmReceive2.cfg and cmReceive3.cfg

To test the Cm data exchange from one process to two or three declared outputs. Notice that cmReceive1 is slow down on purpose to check that it is not blocking cmReceive2. The process cmReceive3 is making itself a connection to the frame sender: cmSend.

This is to test the following keys:

- FDOUT_CM
- FDOUT_CM_TIMEOUT
- FDIN_CM
- FDIN_CM_CONNECT
- FDIN_SLEEP
- FDIN_CM_TIMEOUT
- FDIN_NO_FR_MSG_TIMEOUT
- FDIN_CM_CONNECT

3.2.8. frameMerger.cfg + toFrameMerger_1.cfg and toFrameMerger_2.cfg

This shows an example of a frame merger; to be use together with the toFrameMerger_1.cfg and toFrameMerger_2.cfg as source of frames.
This is to test the following keys:

- FDIN_FRAME_MERGE
- FDOUT_DELAY

3.2.9. playback.cfg

To test the playback mode; You need first to run the simChannels.cfg file.
This is to test the following keys:

- FDIN_PLAYBACK
- FDIN_MIN_LATENCY

3.2.10. softwareInjections.cfg

To test the production of surrogate data and injection of events.
This is to test the following keys

- FDIN_S_INJECTION_FILE

- FDIN_S_INJECTION_EVT_NAME
- FDIN_S_INJECTION_RESCHEDULE
- FDIN_S_INJECTION_RESCALE
- FDIN_S_INJECTION_CHANNEL
- FDIN_ADD_SURROGATE_CHANNEL
- FDIN_ADD_SURROGATE_DQ_CHANNEL

3.2.11. noConfigCheck.cfg:

This is to test the following keys:

- FD_NO_CONFIG_CHECK

3.3. FdGetChannelsList

This program asks the list of channels to a Fd process (FdIOServer or dataSender) and print it. There is only one parameter: the Cm name of the process that we are queering. For instance, to ask the list of channel of the process MainDy, just type:

```
FdGetChannelsList MainDy
```

This program is also an example for the use of the Cm message FdGetChannelsList.

3.4. FdSend

This simple program reads a frame file and sends frames it to a destination. This is more an example and test program than a tool since realistic use requires specifying start time, duration, list of channels, and could be handled by the FdIOServer.

Just type "FdSend" to get the usage.

3.5. FdStat

This program connects to a FdIOServer like server, gets frames from it and displays some statistics about the selected channels.

Just type "FdStat" to get the usage.

3.6. FdWrite

This program writes on disk each frame it receives from a FdIOServer like server. To ask frames to a dataSender, use the FdIOGetFrameFromDataSender program.
The four command line arguments are:

- the target server Cm name,
- the total number of frames/second to write. Zero or a negative value means for ever.
- the file name prefix (like V1:Test)
- the number of frames per file (like 100). Since the boundary of the frame files are integer multiples of the GPS time, therefore the first file is very likely to have less frame than requested.

Example:

FdWrite MainDy 1000 V1:Test 100

Will write 100 seconds of data in 100 seconds long frame file starting by “V1:Test” using frames requested to the “MainDy” process.
Remark:

- The FdWrite process could be stop and any time wit ctrl-C. In this case, the output file is properly closed.
- More complex writing could be made with a FdIOServer.

4. Configuration Parameters

4.1. Configuration access

The following subsections describe the available keywords for the configuration files of a process using Fd. The best example to test these functionalities is the use the generic FdIOServer provided by Fd.
Since Fd is build on top of the Cfg framework, the convention for writing and using the configuration file is the one of Cfg and inherit of the CFG keys.

Remark: by default the parsing step of Fd (function FdIOParse) stops if there are unknown words in the configuration, unless the FD_NO_CONFIG_CHECK is present. Therefore, it is better to enrich the Fd or to disable this option by using this key or setting `fdIO->noConfigCheck = CfgTrue` after FdIONew and before FdIOParseAndIni.

4.2. Keywords to control global parameters

4.2.1. FD_CH_PREFIX

- Parameter 1 (string): To replace the default “V1:Daq” prefix by …

4.2.2. FD_DEBUGLVL

- Parameter 1 (integer): Set the debug level. Set it to 1 or 2 to get some debug information.

This key is intended to be used by experts. It changes the debug level of the following actions. This key could be used multiple times in a configuration

4.2.3. FD_NO_CONFIG_CHECK

This key has no parameters.
If this key is present, the initialization of Fd will not stop if an unknown key is present in the configuration.

4.2.4. FDIN_NO_FR_MSG_TIMEOUT

- Parameter1 (integer): Timeout in seconds before an "no frame read since…” warning message is issued. The default value is 10. The timeout used is this value plus two times the frame duration and is compared to the current time minus the time when the last frame was received. This means that for 1 seconds long frames, this warning message is issued after 10+2x1=12s without receiving a frame.

4.2.5. FDIN_CM_TIMEOUT

- Parameter1 (real): Timeout in seconds when checking for Cm messages at the end of the FdIOGetFrame function. The default value is 0.001 except when reading from file, which set the timeout value to 0. If the timeout is set to a non-zero negative value, the CfgMsgSendWithTimeout is not called.

4.2.6. FDOUT_CM_TIMEOUT

- Parameter1 (real): Timeout in seconds when checking for Cm messages at the end of the FdIOPutFrame function. The default value is 0.0001. Usually, the process is limited by the frame input and therefore only little CPU is waist waiting for new message. If the timeout is set to a non-zero negative value, the CfgMsgSendWithTimeout is not called.

4.3. Keywords to define and control the source of input frames

Only one source of frame could be used. If you application is not generating itself the frames, one, and only one of the above keywords must be used.

- FDIN_CM,
- FDIN_DATA_SENDER,
- FDIN_DIR,
- FDIN_FILE ,
- FDIN_FRAME_MERGER,
- FDIN_NEW_FRAME,
- FDIN_PLAYBACK,

4.3.1. FDIN_CM

This key should be use to accept frame from Cm
It sets the size of the input FIFO when receiving frame via Cm.
This key could not be used with another key defining the input frame like FDIN_FILE

- Parameter 1 (integer): Number of frames which could be buffered in the input Cm buffer. Suggested value: 10. If this buffer becomes full the frames are rejected by the Cm handler until the input buffer is emptied. If set to zero, no frames will be accepted via Cm.
- Parameter 2 (integer): Maximum number of sources which are allowed to send frames to this process. If set this zero, no check on the sources is performed. If it is set to for instance two, only two sources will be accepted. If a third source with a Cm name different from the two first one try to send frames, these frames will be rejected.

4.3.2. FDIN_CM_CONNECT

This key let you specify the frame providers. This key must be added in addition to the FDIN_CM key. This will send the FdAddCmOutput to the appropriate destination.
If a tag has been defined, it is passed along the request, but the FDIN_TAG key must be before the FDIN_CM_CONNECT key.
When the process stop, a message is send to the source to remove the process from the list of destination. However, when using this key for debug, it is advised to set the number of retry is set to zero to ensure that the emission of frame will stop as soon as the process stops, and will not continue after a crash of the server.

- Parameter 1 (char): Cm name of the process on which the connection will be made.
- Parameter 2 (integer): Maximum number of frames which could be in the Cm post output queue of the sending process. If the queue reaches its limit, no more frames are posted, until the queue recover some margin. Suggested value: 5. It this parameter is set to 1, FdIOPutFrame will wait that the frame is successfully send before continuing.
- Parameter 3 (integer): Retry: number of consecutive unsuccessful Cm frame send before the frame output is removed from the list (-1 means output stays forever in the list).

4.3.3. FDIN_DATA_SENDER

This keyword let you connect to a “dataSender”, a remote source of frame send using Cm.

- Parameter 1 (string): Cm name of the dataSender that will send the frames,
- Parameter 3 (integer): GPS time of first data to be read.
- Parameter 4 (integer): Number of seconds of data to be read.

4.3.4. FDIN_DIR

This keyword let you read frame from files located in a directory.

- Parameter 1: (string), directory name,
- Parameter 2: (integer), GPS time of the first file to use.
    - 0 means the latest file. In this case, the directory is scan at start time.
    - -1 means the next file, if the process was already running and then stopped. The needed information is retrieved by reading the information in the file “directoryName/LastFrameInfo-CmName.info” which is provided by a previous run of the application using FdIO. If this file does not exist, the process start at the most recent file.

Remarks:

- Only “.gwf” files contained in the directory will be used.
- File must have the standard name: Prefix-StartGPSTime-duration.gwf, as provided by the FDOUT_FILE key, with only one type of prefix.
- If the process is to slow to read the input directory and is unable to read a file which has just be deleted, it jumps to the most recent file like for a fresh start.
- This key replaces the obsolete FDIN_DIRECTORY.
- This key replaces the old shared memory mechanism by using files in /dev/shm.
- This key could not be used with another key defining the input frame like FDIN_FILE.

4.3.5. FDIN_FILE

This keyword let you take the frames from an input file.
This key could not be used with another key defining the input frame like FDIN_NEW_FRAME

- Parameter 1 (string): Input file name.

- Parameter 2 (integer): Start GPS time (or beginning of the file if this parameter is set to zero or missing). If -1 is used, the start time will the current GPS time (useful for a “playback”).
- Parameter 3 (integer): Number of seconds of data to read (or until the end of the file if this parameter is missing)

Notice that more complex time selection could be made using the SEGMENT tool of the ffl.

4.3.6. FDIN_FRAME_MERGE

- Parameter 1 (int) number of frame to wait for another part
- Parameter 2 (string) list of available accepted sources. The list of expected sources could be updated at run time using Cm messages (see sections 5.1.4 and 5.1.5).

This key implements a frame merger based on GPS time, when receiving frames via Cm. In this case, the function FdGetFrame returns the merged frame. The basics assumptions and the main logic of the frame merger are:

- The list of frame sources is predefined (second parameter of the key). Frames from an unknown source are rejected. Frame sources could be added/removed using the Cm message defined in sections 5.1.4 and 5.1.5.
- The timing difference between the arrival of the frame part from the first source and the one from the last source is limited to a given value, defined in number of frame: FrameDepth (first parameter of the key).
- Frame parts which arrive later than this limit are lost and deleted.
- Once a frame is missing for a source, the frame merger does not wait for this source, until a new frame is received from this source.
- It is assumed that frames coming from a given source arrive always in increasing time. As consequence, an incomplete frame (i.e. with missing source(s)) in the work area of the frame merger will be outputted if a subsequent frame already in the work area is complete.
- The frame duration could be a non integer number of second. It could change from one frame to the next one, although this is an unusual behavior.
- The frame duration should be the same for all sources. In case of difference, the first arrived frame is the reference and the frames with other frame length are deleted.
- The frame merger takes only frame from Cm. Frames received by Cm are put in a FIFO by the Cm handler. This FIFO is emptied by the frame merger before returned the merged frame. If the sources produced burst of frame, to avoid losing frames, it may be needed to increase the size of this FIFO by increasing the value of the first parameter.

The frame merger produces a FrSerData with some general information (number of sources) and the latency of each sources.

4.3.7. FDIN_MAX_SPEED

This keyword let you regulate the frame input speed. This is useful to simulate a frame reading or a frame simulation. This regulation is done at the end of the input frame processing, even if the FD_MAX_SPEED is not the last FDIN key, meaning that if we read 1 second long frame, and ask to resize the frames to 10 seconds long, the regulation will be made on these 10 seconds long frames.

- Parameter 1(float): Minimal number of seconds to wait between two frames reading. To generate “real time”, you should give the inverse of the frame duration like 0.1 for 10 seconds long frames.

4.3.8. FDIN_MIN_LATENCY

- Parameter 1: (integer), Minimal latency requested to accept a frame.

Too early frame are rejected. The default value is -10, even if this key is not part of the configuration file. This means that by default, any frames with a GPS time more than 10 seconds in the future compared to the current time of the machine will be rejected. Notice that the value could be negative to accept “future” frame, like in the case of injection

4.3.9. FDIN_NEW_FRAME

This keyword let you generate new frame header. This is typically used to produce simulated frame with the FDIN_ADD_SURROGATE_CHANNEL key.
This key could not be used with another key defining the input frame like FDIN_FILE

- Parameter 1 (string): Name of the frame to be generated
- Parameter 2 (real): Frame length in seconds
- Parameter 3 (integer): Start GPS time. If a value smaller than 1.e6 seconds is given the start GPS time will the current time delayed by this amount, or in other word, this will be the latency of the frames. If a negative value is given therefore frames will be generated in the future, which is useful for hardware injections for instance. Remark: to generate online frames, you should use at the same time the FDIN_MAX_SPEED to regulate the frame production.
- Parameter 4 (integer): Number of seconds of data to be produce. Zero or a negative value means for ever.
- Parameter 5 (integer): Data Quality flag put in the "dataQuality" word of the frame header.

4.3.10. FDIN_PLAYBACK

This keyword let you take the frames from an input file and "replay" then at a different GPS time. The GPS time of the new frame is updated to the current value.
This key could not be used with another key defining the input frame like FDIN_FILE

- Parameter 1 (string): Input file name.
- Parameter 2 (integer): GPSstart: start GPS time to be used. If this parameter is set to a negative value or zero the file will be read from its beginning.
- Parameter 3 (integer): lenReplay: number of seconds of data to be used. If lenRapley is less or equal to zero, the file is used until its end.
- Parameter 4 (integer): offset when replay de data
- Parameter 5 (integer): number of time the input segment (from GPSstart to GPSstart+lenReplay) needs to be replay. Zero or a negative value means forever.
- Parameter 6 (integer): latency for the production of the playback frames.

Remarks:

- The relationship between the original GPS time (To) and the replayed time (Tr) is:

  $$Ti = Tr - L*(int)((Tr-GPSstar)/ lenReplay)$$

- All input frames should have the same duration which must be an integer number of seconds.
- If the frame production falls behind real time by more than 10 frames, then a bunch a frame is jump to try to recover real time.
- If nReplay >0 the process stop when the input file segment has been read the number of requested times.
- The GPSstart, lenReplay and offset parameters are adjusted to a multiple of the frame length at start time.

4.3.11. FDIN_SLEEP

When FdGetFrame reaches this key, the program goes a sleep according to the given time. This is intended to be use just for library test. .

- Parameter 1 (integer): Number of micro seconds to wait.

4.3.12. FDIN_TAG

This keyword let you select the channels available in the input frame. In the case of reading from file (FDIN_FILE, FDIN_DIR and FDIN_PLAYBACK) only the relevant channels are read which could speed up the process. For FDIN_CM, the channel selection is performed after the frame is received (unless the FDIN_CM_CONNECT key has been used). For FDIN_DATA_SENDER, the tag is propagated to the frame source and the frames are sent using this tag.

- Parameter 1: (string): Selection of channels on the input frames.

4.4. Keywords to control the input frames content

4.4.1. FDIN_ADD_SURROGATE_CHANNEL

This will create a channel with a spectral density given by the vector stored a priori or with a normal (Gaussian) distribution.

- Parameter 1 (string): Name of the new channel.
- Parameter 2 (string): Unit.
- Parameter 3 (double): Sampling rate in Hz.
- Parameter 4 (integer): Number of bits used to store the results. The allow values are:
  - 32 for 32 bit integer,
  - 16 for 16 bits integer,
  - -32 for 32 bits floating point,
  - -64 for 64 bits floating point.
- Parameter 5 (string): File name which contains the vector or keyword "NORMAL" to generate a Gaussian distribution with a spectral density given by the scaling factor (parameter #6). I the spectral density is given by a vector, two types of vector are allowed;
  - FrVect (file extension type ".vect").
  - ASCII file (file extension type ".txt"). In this case, each line of the file should contain the frequency and the corresponding amplitude. Additional characters are allowed on each line after these to values.
- Parameter 6 (double): Scaling factor if data are store as integer. If not present, this is set to 1.
- Parameter 7 (integer): Seed for the random number generator. If not present, this is set to 0. Each surrogate channel has its own random number sequence, which means that adding another channel, is not changing the sequence of random numbers.

The data are generated by first producing a vector which contain white noise, and then apply a frequency domain filtering. The length of the FFT used is twice the duration of the input frame. Don't forget that strange effect could be observed if the dynamic of the spectrum is large (more than 5 order of magnitude). In that case, increasing the FFT length (by increasing the input frame length) is recommended.

4.4.2. FDIN_ADD_SURROGATE_DQ_CHANNEL

This will create a simulated data quality channel. This is a channel which can take two values: a true and a false value. The duration of the false segments have a random flat distribution with a minimal and maximum duration and a total duty cycle (for the true value). This channel is stored as a 4 bytes signed integer FrAdcData channel.

- Parameter 1 (string): Name of the channel.
- Parameter 2 (double): Sampling rate in Hz.
- Parameter 3 (integer): false value like 0.
- Parameter 4 (integer): true value like 12.
- Parameter 5 (integer) : minimal segment length in seconds
- Parameter 6(double): maximum segment length in seconds
- Parameter 7 (double) duty cycle for the "true" value.
- Parameter 8 (integer): Seed for the random number generator. If not present, this is set to 0. Each surrogate channel has its own random number sequence, which means that adding another channel, is not changing the sequence of random numbers.
- Parameter 9 (string): Name of the h(t) channel to be reset to zero if the DQ channel is false. Put a non existing channel name to do nothing.

4.4.3. FDIN_ADD_TEST_CHANNEL

This will create a FrAdcData channel filled with a sine wave plus a Gaussian noise.

- Parameter 1 (string): Name of the new channel.
- Parameter 2 (string): Unit.
- Parameter 3 (double): Sampling rate in Hz.
- Parameter 4 (integer): Number of bits used to store the results. The allow values are:
  - 32 for 32 bit integer,
  - 16 for 16 bits integer,
  - -32 for 32 bits floating point,
  - -64 for 64 bits floating point.
- Parameter 5 (double): mean value for the white noise

- Parameter 6 (double): RMS for the white noise
- Parameter 7 (double): sin amplitude
- Parameter 8 (double): sin frequency
- Parameter 9 (double): Scaling factor. If not present, this is set to 1.

The main purpose of this key is to produce test channels for testing the Fd library and related applications.

4.4.4. FDIN_BUT_FILTER

This key applies low path Butterworth filters when reading the frame. New channels are created with names containing the suffix that gives the decimation frequency (for instance for Pr_B1_ACp filter with a decimation frequency of 50Hz, the channel Pr_B1_ACp_50Hz is created). If the input channel already has a frequency suffix, this suffix is replaced by the new one in order to avoid double or more frequency suffixes. More information about the filter used is recorded in the "comment" field of the created channel.

- Parameter 1 (integer): Order of the filter.
- Parameter 2 (string): Tag that defines the channel on which the filters are applied.
- Parameter 3 (double): Cut off frequency for the filter, usually a little less than half the new sampling frequency.
- Parameter 4 (double): New sampling frequency. After filtering the frequency of the channel could be reduced. In that case, one sample (the last) out of freqIn/freqNew is taken. .
- Parameter 5 (integer): An optional parameter to control the filtering process:
    - 0 (default value) means original channels are deleted, and the type of the created channel is defined according to the following logic: 32 and 64 bits floating points remains unchanged, all integer are converted to 32 bits integers.
    - 1 mean keep the input channels in addition to the filtered channels. Same logic as above for the type of the filtered channels.
    - 2 means the original channels are deleted and all created channels are 64 bits floating points.
    - 3 means keep the input channels in addition to the filtered channels and all created channels are 64 bits floating points.

Remarks:

- Channels with a sampling rate equal or smaller than the requested new sampling rate are not processed.
- If the request new sampling rate is not an integer divider of the channel rate, the new sampling rate for this channel is the largest possible value just above the requested value (example: if 50Hz is request for a 120Hz channel, the produced frequency will be 60Hz).
- This filtering propagates the auxiliary vector which indicate missing sample, if this information is present for the input vector.
- Multiple FDIN_BUT_FILTER keys could be use. They are executed according the sequence of FDIN keys. They may use channel produced by the previous FDIN key.

4.4.5. FDIN_COMBINE_CHANNELS

This key let you combine two channels, creating a new one. The new channel is

output = scale1 * channel1 + scale2 * channel2

with

- Parameter 1 (string): name of the output channel. It could be one of the input channels.
- Parameter 2 (double): scale1 coefficient.
- Parameter 3 string): channel1 name.
- Parameter 4 (double): scale2 coefficient.
- Parameter 5 string): channel2 name.

Remark: the output channel is created with the type double, unless both input channel are of type float (4 bytes).

4.4.6. FDIN_COMPRESSION

- Parameter 1 (integer): It set the frame compression level for the frame reading from file or frame received by Cm. The compression flag values are the ones defined in the frame library (Fr package). The default value is -1 (no action done) if this key is not present. Other useful values are 0 (uncompress data) and 9 for optimal compression.

Remarks:

- This key must be placed before the input frame file is defined.

- When compressed frame is used and an action is performed on channel like a low pass filter, the data of this specific channel are decompressed when performing the action and the untouched channels remain compressed.

4.4.7. FDIN_DUPLICATED_CHANNELS_CHECK

If this keyword is present, the input frame is scan for ADC, serData and Proc data duplicated channel(s), i.e. two or more channels with the same name.

- Parameter 1 (integer): Size (in bytes) of the buffer used to print the list of the first duplicated channels. The default value is 200 bytes . This is an optional parameter.

Remarks:

- The number of duplicated channel is stored in the process FrSerData with the name nDuplicatedChnl.
- if more than one check is requested as FDIN and FDOUT, the additional check(s) will report the number of duplicated channels in the variable name nDuplicatedChnlN with N being the instance number.
- If duplicated channels are detected, the status of this Fd object is changed from "Golden" to "Active"
- This performs only a check. To remove the duplicated channel, instead of using this key you should used the FDIN_DUPLICATED_CHANNELS_REMOVE key.

4.4.8. FDIN_DUPLICATED_CHANNELS_REMOVE

With this key the same check as for FDIN_DUPLICATED_CHANNELS_CHECK are performed. In addition, only one instance of the duplicated channels is kept for Adc and Proc data, the other are removed.

- Parameter 1 (integer): Size (in bytes) of the buffer used to print the list of the first duplicated channels. The default value is 200 bytes. This is an optional parameter.

Remarks:

- The number of duplicated channel is stored in the process FrSerData with the name nDuplicatedChnl.
- if more than one check is requested as FDIN and FDOUT, the additional check(s) will report the number of duplicated channels in the variable name nDuplicatedChnlN with N being the instance number.
- If duplicated channels are detected, the status of this Fd object is changed from "Golden" to "Active"

4.4.9. FDIN_FFT_FILTER

This apply frequency domain filtering when reading the frame. New channels are created with names containing the suffix that gives the high pass frequency (if different from zero) and the new sampling frequency which is twice the low pass frequency. For instance for Pr_B1_ACp filter with a high pass filter of 10Hz and a new sampling frequency of 4096Hz, the channel Pr_B1_ACp_10Hz_4096Hz is created).

- Parameter 1 (string): Tag that define the channel on which the filters are apply
- Parameter 2 (double): Frequency for the high pass filter. Zero is a valid number, but a DC offset might be observed in the output signal.
- Parameter 3 (double): New sampling frequency; the low pass frequency is half of this number. This value could be higher than the original sampling rate. In this case, the signal is re-sampled at higher frequency.
- Parameter 4 (double, optional): Delay applied when filtering the channel.

Remarks:

- The FFT are performed on the size of the frame. Therefore, this key introduces a latency of one frame in the FDIN pipeline.
- Multiple FDIN_FFT_FILTER keys could be use. They are executed according the sequence of FDIN keys. They may use channel produced by the previous FDIN key.

4.4.10. FDIN_FRAME_DURATION

- Parameter 1(float): New length in seconds of the input frames. The input frame length could be change from rather arbitrary values like from 1 to 10seconds, from 10 to 0.2 seconds or 16 to 10 seconds long.

4.4.11. FDIN_NOMSG

- If this key is present, the FrMsg are removed from the input frames

4.4.12. FDIN_RANGE_GATING

This let you compute an inspiral range and has to option to apply a gating on the h(t) channel.

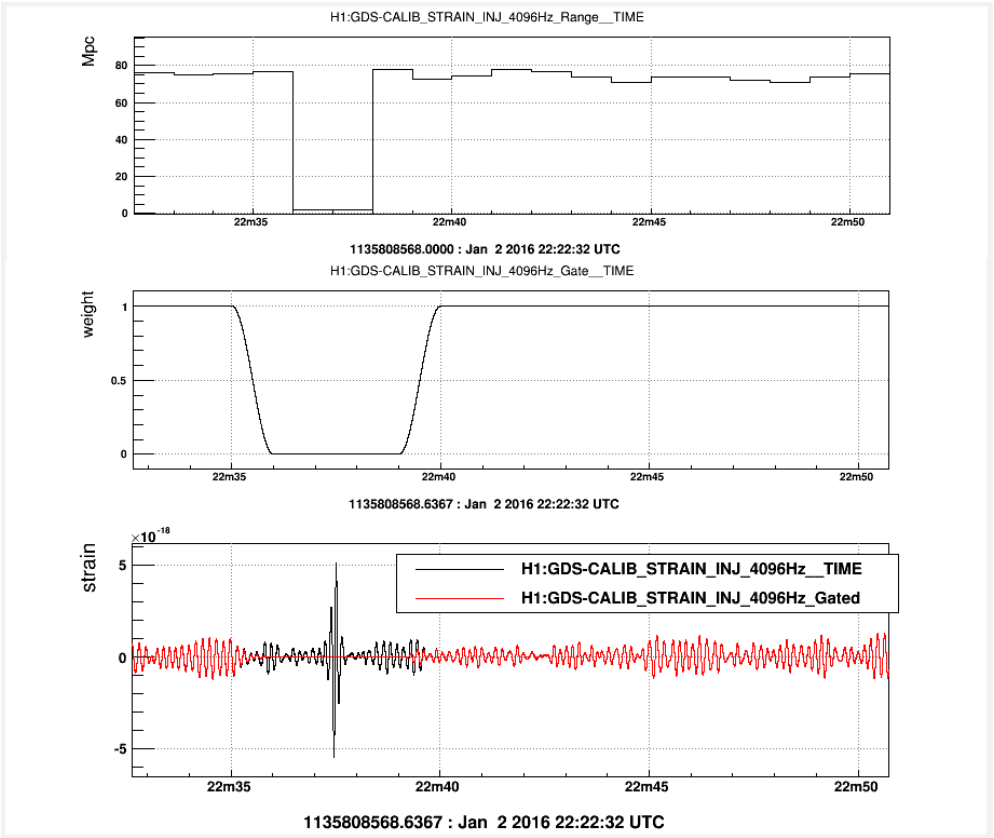
- Parameter 1 (string): name of the h(t) channel to be monitored/gated.
- Parameter 2 (real): mass of the single object used to compute the inspiral range. Use 1.4 for BNS.
- Parameter 3 (int): nChunk: number of time the frame length is divided to get the FFT duration. Suggested value: 4.
- Parameter 4 (real): range threshold. When the range is below this value, h(t) is set to zero. If this parameter is set to zero or missing, the range is computed but no gating is applied.
- Parameter 5 (int): if different from zero, the h(t) gated data are put in a new channel (adding the suffix "_Gated" to the h(t) channel name).

Remarks:

- This function computes the inspiral range on one frame long. Use the FDIN_FRAME_DURATION key to compute the range over a give time. The FFT are computed over the frame duration divided by nChunk (the third parameter), with an overlap of 50% with the next FFT. This means that 2*nChunk -1 FFTs are averaged to get h(t) spectrum used for the inspiral range computation.
- If the range is bellow a threshold, a gating is applied on a frame length plus two windows of half a frame each. This is why the code is working with two frames, in order to take care of the glitches at the overlap and apply gating with a window.
- The gate function goes from 1 to 0 (and vice versa) over half a frame using half of a sine wave function.
- This key introduces a latency of one frame in the pipeline.

Example:

The following figure is showing the time series for the inspiral range of H1 data (top plot), the weigth apply on the h(t) time serie (middle plot) and on the bottom plot the raw h(t) (black curve) together wigth the gated h(t) (red curve). A loud glitch is producing a drop of the inspiral range which is computed on two second long frames. The drop is obsertved on two bins of the range plot (these bins are one seconds long but due to FFT overlap, this meanse that the glitch could be within the three seconds used to compute these two range. This is why the gate function is set to zero during 3 seconds.



4.4.13. FDIN_RESCALE_CHANNEL

This let you rescale on FrAdcData or FrProcData channel, to change for instance the channel unit. The type of the vector output is the same as one of the input vector.

- Parameter 1 (string): Name of the output channel. This channel is created unless it is the same as the input channel in which case, the input channel is rescaled.
- Parameter 2 (real): Scaling factor
- Parameter 3 (string): Name of the input channel.
- Parameter 4 (string): New unit (optional parameter)

4.4.14. FDIN_STAT

- Parameter 1 (string): This is the suffix added to the name of the statistics. If this parameter is missing, no suffix is added.

This key enable the computation of frame statistic like the number of ADC channel (variable name: nAdcSuffix were "Suffix" is the parameter of this key), the number of SerData (nSms), the number of proc channel (nProc), and the latency. Some FD keys are also computing summary information which are added to the FrSerData.
All these variables are stored in a FrSerData named as Daq_processNameSuffix (where processName is the configuration name and suffix the only parameter of the FDIN_STAT key) and added to the frames.
Running time information is also provided as a "UserInfo" message which is display in the VPM GUI for instance.
It is not possible to have multiple FDIN/OUT_STAT key with the same suffix.

Usually we expect only one FDIN/OUT_STAT but multiple instances with different suffix could be used for better reporting.

These statistics are computed when the FdIOGetFrame function reaches this key. Therefore, if there is some processing, the results depend on the location of the key. It is therefore likely that some of the statistics, especially the output statistics are given for the previous frame.

If an application requires do build its own UserInfo message, the developer could use the function FdStatBuild (see section 2.2.6) to get the Fd information without using the FDIN_STAT key.

4.4.15. FDIN_RECAST_CHANNELS

This object changes the vector type with a possible rescaling. This works only for the FrProcData and FrAdcData structures. The input vector could only be of type 4 or 8 bytes floating point and 2 or 4 bytes integer.

- Parameter 1 (string): A tag which defines the channel or group of channels going to be recast. This could be a single channel name, or a group of channels using the usual FrameLib convention ("V1:Pr* V1:Em*" will process all channel with name starting by "V1:Pr" or "V1:Em").
- Parameter 2 (int): nbits (int): number of bit for the new vectors. Values could be:
  - -64 to go to 8 bytes floating points (FR_VECT_8R)
  - -32 to go to 4 bytes floating points (FR_VECT_4R)
  - 16 to go to 2 bytes integers (FR_VECT_2S)
  - 32 to go to 4 byte integers (FR_VECT_4S)
- Parameter 3 (int): "mode" describing how the creation: deletion of new channel(s) and channel name(s) is handled. Value could be:
  - mode = 0: Channel(s) content will be replaced
  - mode = 1: Create new channel(s) with suffix ("_8R", "_4R", "_2S" or "_4S" is added)
  - mode = 2: Create new channel(s) and add suffix to old channels.
  - Remark: if the input channel has already the right type and scale = 1, then nothing is done.
- Parameter 4 (double): The scaling factor applied when converting the vector. This is useful when converting to integer to match the dynamic of the signal to the integer range.
  - When applied on the ADC channel, the overall unit is preserve (the ADC "slope" is updated)
  - When applied on ProcData channel, the scale is recorded in the comment.
  - If the scaling factor is zero, then for ADC channel, the scaling factor used is the adc->slope value, which becomes 1 after this recast. This is useful when we don't want to use slope value on the resulting vector.

Remark: when converting a channel with floating point values to integer values, the numerical noise introduce is 1./(scale*sqrt(12*sampleRate)) where "scale" is the applied scaling factor and "sampleRate" the sampling frequency of the channel.

4.4.16. FDIN_REJECT_EVENTS

This key let you reject events on the basis of event parameters.

- Parameter 1 (string): tag used to select the events based on their names. For instance, if the tag is "MbtaH* MbtaL*", the rejection criteria will be applied on all events with a name starting by MbtaH or MbtaL.
- Parameter 2 (string): Parameter used for the selection. It could be any event parameter or the event amplitude (use in this case "amplitude").
- Parameter 3 (double): min value used for the selection.
- Parameter 4 (double): max value used for the selection.
- Parameter 5 (int): down sampling factor. This is the reduction factor applied on the selected events, keeping one event out of the down sampling number. If set to zero or if this parameter is missing, all events which fulfill the selection are reject.

Examples:

- FDIN_REJECT_EVENTS "Mbta*" H1:SNR 0 5 10 will reject 9 out of 10 events with a name starting by Mbta and 0 <= H1:SNR < 5

Remarks:

- Multiple FDIN_REJECT_EVENTS key could be used.
- The down sampling parameter is added to the event parameters for the events which fulfill the selection criteria but have been kept if the down sampling is used instead of just a simple rejection.
- If more than one down sampling is used, like using the keys
  FDOUT_REJECT_EVENTS "*" amplitude 0 5.5 10
  FDOUT_REJECT_EVENTS "*" amplitude 0 6.0 10, the product of the two down sampling factor is recorded for the applicable triggers, in this case a down sampling factor of 100 for triggers up to an amplitude of 5.5

4.4.17. FDIN_RENAME_CHANNEL

This object changes the name of a channel (FrAdcData or FrProcData), without changing its type or content.

- Parameter 1 (string): The old name. This must be a single name. No wild card could be used.
- Parameter 2 (string): The new name

4.4.18. FDIN_SELECT_CHANNELS

This key let you reject channels, with the possibility to periodically keep them for a limited time.

- Parameter 1 (string): tag used to define the channels which are always kept. Channels with names not matching the tag will be rejected, except during the time define by the two following parameters. For instance, if the tag is "* -Pr*", all channels except the channels starting by Pr" will be selected.
- Parameter 2 (int): Period used to keep the rejected channels. If this parameter is set to zero or missing, the channel selection is always applied.
- Parameter 3 (int): time window used to periodically keep the rejected channels. If this parameter is set to zero or missing, the channel selection is always applied.

Examples:

- FDIN_SELECT_CHANNELS "* -*tmp" 1000 10 will remove all channels finishing by "tmp" except for all frames with a GPS time ending by 000, 001, … 009.

4.4.19. FDIN_S_INJECTION_FILE

With this key a software injection file will be read. Only one injection file (.gwf or .ffl) could be used. The injection file must contain the simulated channel(s) and the FrSimEvent structure describing it. Notice that you need to use at least a FDIN_S_INJECTION_CHANNEL key to define on which channel the simulated signal will be added.

- Parameter 1 (string): The name of the file which contains the software injection.
- Parameter 2 (integer): start time to read the file, if we do not want to read the full file. Zero means start at the file begin.
- Parameter 3 (integer): duration of the input file to read. Zero means use the full file.

4.4.20. FDIN_S_INJECTION_EVT_NAME

This key defines the name(s) of the FrSimEvent to be searched in the injection file. With the signal is injected around the injection, when the simulated signal is non zero. It is then possible to reschedule the injection multiple times (see the FDIN_S_INJECTION_RESCHEDULE key). Without this key, the segment defined by the parameter 2 and 3 of the FDIN_S_INJECTION_FILE key is injected.

- .Parameter 1 (string): name(s) of the FrSimEvent. Multiples names using wild card could be used like "C* B*" to search for all events with a name starting with a B or C.

Remark: The use of this key requires a preliminary scan of the injection file which might be long if there are many events in the file.

4.4.21. FDIN_S_INJECTION_RESCHEDULE

This key let rescheduled the software injections at a different time. Without this key, the events are injected at their own time. It must follow a FDIN_S_INJECTION_EVT_NAME key.

- Parameter 1 (double) period (in seconds) between two consecutive set of injections if we want to replay the injections more than one. If this parameter is set two zero (or missing as well as the following parameters), the injections is made only once.
- Parameter 2 (double) maximum time jitter (in seconds) applied when scheduling an event injection. The effective time jitter is uniformly distributed between 0 and plus or minus the maximum time jitter.
- Parameter 3 (double) time offset (in seconds) when computing the first injection. This could be a negative value like minus half of the injection period.

4.4.22. FDIN_S_INJECTION_RESCALE

This key let rescaled the software injection with different amplitude. It must follow a FDIN_S_INJECTION, FDIN_S_INJECTION_CHANNEL or FDIN_S_INJECTION_RESCHEDULE key.

- Parameter 1 (real): scaleMin: minimal rescaling factor to rescale the injection. The scaling factor applied to the injection amplitude is uniformly randomly distributed between scaleMin and scaleMax where scaleMax is the next parameter. If this parameter is set to zero (or missing as well as the following parameters), no rescaling is applied.
- Parameter 2 (real): scaleMax: maximum rescaling factor. If this parameter is missing or less than scaleMin, all injections are rescaled with the same factor given by scaleMin.

Remark: if the injections are played only once (i.e. they are not rescheduled or replay using the keys FDIN_S_INJECTION_EVT_NAME or FDIN_S_INJECTION_RESCHEDULE) the rescaling factor is just the first parameter, the second parameter is not used.

4.4.23. FDIN_S_INJECTION_CHANNEL

This key defines on which channel the software injection should be added.
This must be the last of the FDIN_S_INJECTION key: the file and event name must be defined before.
We could have multiple channels defined if we want coherent injections on multiple streams.

- Parameter 1 (string): name of the simulated waveform channel
- Parameter 2 (string): name of the channel on which we add the injection
- Parameter 3 (string): name of the FrSerData structure to be copied in the output file when reaching their time. The value could be "*". This third parameter is optional. It is useful and activated only when the FDIN_S_INJECTION_EVT_NAME key is not used.

4.4.24. FDOUT_SET_PROC_TYPE

This key let you set the type of a group of FrProcData structure.

- Parameter1 (string): Tag to select the channel(s) for which the type is set (like "*AC*".
- Parameter2 (dec): New type value.

4.4.25. FDIN_FSHIFT

This object applies a frequency shift. This could be use also to add a fixed or variable phase offset.

- Parameter 1 (string): Output channels name; without the "p" or "q" suffix. The output type is a FrAdcData structure to keep track of the applied phase shift.
- Parameter 2 (string): Input channels name; without the "p" or "q" suffix. This works only for FrAdcData structures.
- Parameter 3 (real): Frequency shift in Hz. Zero is a valid option.
- Parameter 4 (string): Phase Shift in rad. If it starts with a digit, the string is interpreted as a fixed floating point value. If not, this is the name of the channel (ADC or Proc) containing the phase offset which could be time dependant.

The output signals are computed as:

$$OUTp(t) = INp(t) *\cos(2\pi frequency*t + phase(t)) - INq(t)*\sin(2\pi frequency*t + phase(t)))$$
$$OUTq(t) = INp(t) *\sin(2\pi frequency*t + phase(t)) + INq(t)*\cos(2\pi frequency*t + phase(t)))$$

Where t is the GPS time.

4.4.26. FDIN_DEMOD

This object demodulated a signal.

- Parameter 1 (string): Output channels name; without the “p” or “q” suffix. The output type is a FrAdcData structure to keep track of the applied phase shift which is stored as metadata of the FrAdcData structure. The type of the output vector(s) is a 64 bits double.
- Parameter 2 (string): Input channel name; It should be a FrAdcData structures.
- Parameter 3 (real): Demodulation frequency in Hz
- Parameter 4 (string): Phase Shift in rad. If it starts with a digit, the string is interpreted as a fixed floating point value. If not, this is the name of the channel (ADC or Proc) containing the phase offset which could be time dependant. The phase is given for t=0.
- Parameter 5 (integer) sampling frequency in Hz of the output signal.
- Parameter 6 (real) 3dB frequency cut of the output low pass filter.
- Parameter 7 (integer) order of the output filter. 0 means simple decimation.

The output signals are computed as:

- OUTp(t) = LowPassFilter(2*Input(t) *cos(2πfrequency*t + phase(t)))
- OUTq(t) = LowPassFilter(2*Input(t) *sin(2πfrequency*t + phase(t)))
- Where t is the GPS time.

4.4.27. FDIN_GET_PHASE

This object extracts the phase by taking the arc tan of two signals.

- Parameter 1 (string): Name of the output phase signal. The output signal is an FrAdcData of type double in radian.
- Parameter 2 (string): Name of the input channel without the suffix “p” and “q”. They should be of type FrAdcData.
- Parameter 3 (real): frequency shift in Hz. Zero is a valid option.
- Parameter 4 (string): Phase shift correction in rad. If it starts with a digit, the string is interpreted as a fixed floating point value used to correct for the arbitrary phase. If not, this is the name of the channel containing the phase offset which could be time dependant.

The output signal is computed as:

- Out(t) = atan2(INp(t), INq(t)) - phase(t);

4.4.28. FDIN_GET_MODULUS

This object extracts the modulus by taking the modulus of two signals
This works only for FrAdcData structures. The output signal is a FrProcData of type double (or float, depending on the required storage.
The output vector is stored as a double precision number
The input vector could only be of type 4&8 bytes floating point and 2&4 bytes integer.

- Parameter 1 (string): Name of the output phase signal. The output signal is an FrProcData of type double in radian.
- Parameter 2 (string): Name of the input channel without the suffix “p” or “q”. This could be an FrProcData or FrAdcData structures. The vector could only be of type 4&8 bytes floating point and 2&4 bytes integer.

The output signal is computed as:

- Out(t) = sqrt(INp(t)**2, INq(t)**2);

4.5. Keywords to control the frame output(s)

4.5.1. FDOUT_BUT_FILTER

Same object as FDIN_BUT_FILTER, but applied at the output time.

4.5.2. FDOUT_DUPLICATED_CHANNELS_CHECK

Same object as FDIN_DUPLICATED_CHANNELS_CHECK, but applied at the output time.

4.5.3. FDOUT_DUPLICATED_CHANNELS_REMOVE

Same object as FDIN_DUPLICATED_CHANNELS_REMOVE, but applied at the output time.

4.5.4. FDOUT_CLEAN_DIR

- Parameter 1 (string): Name of directory where to clean up the too old frame files.
- Parameter 2 (integer): Period in seconds for doing the cleaning
- Parameter 3 (integer): Maximum number of files to keep or zero if not used as a criteria
- Parameter 4 (real): Maximum space (in GB) used by all files or zero if not used as a criteria
- Parameter 5 (integer): Maximum time span (in seconds) used by all files or zero if not used as a criteria

A full scan of the directory is performed at each period to have an update list of available frame files (i.e. of type .gwf), and to account for files which might have been removed manually. This scan determines the number of frame files, the time span covert by the files (information are extracted by the standard .gwf file names) and the disk space used. If there are too many files the oldest files are removed to keep the directory with the limits set. If multiple limit are set, like number of files and disk space, files are removed until all criteria are fulfilled.

The directory is expected to contain only one type of frame files with the same file prefix. If it is not the case a non fatal error message is issued.

This key is indented to make a circular buffer of frame in connection with the use of the FDOUT_FILE key. Example:

```
FDOUT_FILE      /dev/shm/FdTest/V  1 "fastAdc1"
FDOUT_CLEAN_DIR /dev/shm/FdTest/ 10 0 100 1
```

These two keys create a circular buffer in the /dev/shm/FdTest directory of no more that 100 seconds long and using no more than 1GB.

Remark:

- This key replaces the obsolete FDOUT_CLEAN_DIRECTORY key.
- If the three limits are set to zero, no cleaning is performed but the ffl could be build

4.5.5. FDOUT_CLEAN_DIR_FFL

- Parameter 1 (string): name of the ffl to be created.
- Parameter 2 (string): the name of another ffl (or file) to be included at the beginning of the produced ffl. This is an optional parameters. It allows producing ffl made of two directories.
- Parameter 3 (int): maximum time overlap between the included ffl and the first (oldest) file of the directory. This is useful when do not want that the included ffl hide a big fraction of the files of the directory when they are overlapping. Remark: the overlap should be a multiple of the frame length, but it is up to the user to pay attention to this.

With this key an ffl is created and updated after each scan of the directory by the FDOUT_CLEAN_DIR key.
This key must be placed after the FDOUT_CLEAN_DIR key which defines the directory on which the ffl will be build and updated after each files change.
If the included ffl is covering a shorter time range than the files of the directory, this ffl is not included in the resulting ffl.

4.5.6. FDOUT_CM

Defines the Cm output(s):

- Parameter 1 (string): Cm name of the frames destination.
- Parameter 2 (string): Tag for channels selection. It could be the name of one channel like "Pr_B1_DC", it could be a group of channels like "Em_AC*" (all channels with a name starting by Em_AC) or it

could be all the channels: "*". If the string contains one of the keywords \#ADC, \#SER or \#PROC, any other selection is invalidated. For instance "Pr_B1_ACp \#ADC Em* \#SER -*Alp*" sends frames containing only AdcData channels whose name begins with Em and SerData channels whose name does not contain the word Alp. Pr_B1_ACp selection is cancelled.

- Parameter 3 (integer): Maximum number of frames which could be in the Cm post output queue. If the queue reaches its limit, no more frames are posted, until the queue recover some margin. Suggested value: 5. It this parameter is set to 0, FdIOPutFrame, do not use the post mechanism but just the CmMessageSend which means it will wait that the frame is successfully send before continuing.
- Parameter 4 (integer): Retry: number of consecutive unsuccessful Cm frame send before the frame output is removed from the list (-1 means output stays forever in the list).
- Parameter 5 (integer): checksum computation. If this parameter is missing or set to zero, no checksum is computed when sending the frame via Cm. If the first bit (i.e. like 1 or 3) is set: the file checksum is computed. If the second bit is set (i.e like 2 or 3) : the structures checksum are computed. To compute all checksums, use “3”.

Remark:

- Multiple FDOUT_CM keys could be use at the same time with different destinations
- Since version 8 of Fd, thanks to the output queue, the emission of frame is done asynchronously. This means that if a process sends frames to two destinations and one of them is very slow, this does not impact the other destination.
- If the queue becomes full, the emission of the frame is stopped until the queue is emptied.

4.5.7. FDOUT_CM_SERVER

This key indicates where a process could connect. It also takes care of keeping a running list of channels to be distributed to processes like dataDisplay making the appropriate request, which could be send even if the process is waiting for a new frame..

- Parameter 1 (integer): Maximum number of process which could ask for frames via Cm. Zero is a valid answer which means that no Cm connections are accepted and the process will not answer to the FdGetChannelsList Cm request.
- Parameter 2 (integer): list lifetime: time to keep a channel in the list of channel after it is gone. If this is set to zero, the list of channel is only build on request when passing through this key.
- Parameter 3 (integer): If this parameter is present and larger than zero, it defines the Cm output queue size and the Cm post mechanism with this queue size will be use for all connecting processes, regardless of parameter provided in their request.

Remarks:

- There could be only one FDOUT_CM_SERVER key per process.
- If no FDOUT_CM_SERVER key are in the configuration, a default one is added at the end of FdIO initialization with a maximum number of 5 Cm connections and the list lifetime set to zero, meaning that the list of channel is only build when FdIOPutFrame reaches this key.
- A side activity of building the list of channel (i.e. lifetime > 0) is to monitor the number of channel created, removed and duplicated. Information message are provided in the log file and as USERS_INFGO. Therefore it is advised to have the lifetime large enough to not generated too many message in the log files.

4.5.8. FDOUT_COMBINE_CHANNELS

Same as FDIN_COMBINE_CHANNELS but applied at output stage.

4.5.9. FDOUT_COMPRESSION

- Parameter 1 (integer): It set the frame compression level. The possible values are the ones of the frame library (Fr package). The default value is -1 (no action done) if this key is not present. Other useful values are 0 (uncompress data) and 9 for optimal compression.
- Parameter 2 (integer): number of thread created for the compression. This is an optional parameter. If it is not present or if zero is given, no threads are created. Otherwise, this is a good solution to parallelize the compression step which could be time consuming for large frames.

Remarks:

- In the case of the frames send over cm, the frame checksums are not computed. To activate the checksums computation, set the compression value to a negative value. For instance -9 will compute the

checksum and will use the frame compression type 9. In the case of frames written on disk, checksums are always computed.
- The compression is performed when this key is reached. If channels are added after, they will not be compressed

4.5.10. FDOUT_CONSUMER

This is an obsolete keyword replaced by FDOUT_CM

4.5.11. FDOUT_CONVERT_SERDATA

This triggers the conversion of all FrSerData to FrAdcData. It has no parameters.

4.5.12. FDOUT_DELAY

This key should be use to delay (buffer) the frames before outputting them.
- Parameter 1 (integer): The delay express as number of frame. Should be multiply by the frame length to get the effective delay.

4.5.13. FDOUT_FFT_FILTER

Same object as FDIN_FFT_FILTER, but applied at the output time.

4.5.14. FDOUT_FILE

This key defines output on files. Multiple FDOUT_FILE could be used to write different stream in parallel.
- Parameter 1 (string): Path and name prefix of the files where frames will be written. The GPS time, file duration and ".gwf" file suffix are automatically added to the file name.
- Parameter 2 (integer): File duration in seconds. A new file is open each time the GPS time of the frame reach a multiple of this number. Therefore the file start is aligned on a multiple of this number, except for the first file which start at the first frame produced and could be shorter.
- Parameter 3 (string): Tag: a selection of channels to be kept in the output frames (see the FDOUT_CM key for the tag definition).
- Parameter 4 (integer): This optional parameters lets define period (in seconds) of the directories. When this period is reach, a new directory is open. If the parameters 1 is "/somewhere/V1", then the directories are named "/somewhere-GPS" with GPS being the GPS value of the first frame in the directory and the values are named V1-GPS.gwf. If this period is set to zero (or the parameter not present) this feature is not enabled.

Remarks:
- Write errors will change the state of the program to CfgServerError
- The directory must exist (if not, the program stop with a fatal error) unless the output directory is in /dev/shm, in this case, the directory is created (this is to avoid problem after a reboot of a machine which remove all files in /dev/shm).
- The first parameter should not end with a "/" in order to be able to extract a prefix for the file name.

4.5.15. FDOUT_FREQUENCY_CUT

With this key, all channels with a sampling rate faster than requested will be removed.
- Parameter 1 (string): Tag: a selection of channels to be kept in the output frames (see the FDOUT_CM key for the tag definition).
- Parameter 2 (double): maximum frequency allows.

4.5.16. FDOUT_FRAME_DURATION

Same object as FDIN_FRAME_DURATION, but applied at the output time.

4.5.17. FDOUT_RECAST_CHANNELS

Same object as FDIN_RECAST_CHANNELS, but applied at the output time.

4.5.18. FDOUT_RENAME_CHANNEL

Same object as FDIN_RENAME_CHANNEL, but applied at the output time.

4.5.19. FDOUT_REJECT_EVENTS

Same object as FDIN_REJECT_EVENTS, but applied at output time.

4.5.20. FDOUT_RESCALE_CHANNEL

Same object as FDIN_RESCALE_CHANNEL, but applied at the output time.

4.5.21. FDOUT_SELECT_CHANNELS

Same object as FDIN_SELECT_CHANNELS, but applied at the output time.

4.5.22. FDOUT_SET_NEXT_STOP

This optional key, which has only one integer parameter, lets redefine the default period (1 second) the process wait before stopping on request of the *FdSetNextStop* cm message (see5.1.6). This key is useful when the frames are extended. In this case, it is suggested to put the same value as the new frame duration to avoid stopping with an incomplete frame.

4.5.23. FDOUT_SET_PREFIX

- Parameter1 (string): Prefix to be added to the name of each channel of the output frames (like: "V1:").
- Parameter2 (string): Tag to select the channel(s) on which the prefix is added (like "*AC*".

With this key the prefix defined has parameter 1, will be added on all channels selected by the tag defined as parameter 2. The prefix is not added for the channels already starting by the requested prefix.

4.5.24. FDOUT_SET_PROC_TYPE

Same object as FDIN_SET_PROC_TYPE, but applied at the output time.

4.5.25. FDOUT_STAT

Same object as FDIN_STAT, but applied at the output time.
Remarks:

- The uniqueness of the Suffix for the statistics name is valid jointly for the FDIN_STAT and FDOUT_STAT keys.
- The statistic for the frame output like the number of MB written are put in the following frame

4.5.26. FDOUT_TREND_FRAME

This key enables the production of trend frame. A trend frame capture the min, max, mean and rms values of all FrAdcData and FrProc channels over one seconds. FrSerData are converted to FrAdcData channels. The output frame is replaced by the trend frame..

- Parameter 1 (string): A "tag" defining the parameter on which a the trend frame is build
- Parameter 2 (integer): The trend frame length in seconds

Remark: it is possible to write a regular file and a trend file in the same process by putting first a FDOUT_FILE key to write the regular file, then the FDOUT_TREND to produce the trend frame and then another FDOUT_FILE to write the trend frame.

5. Cm messages available

These sections list the Cm messages which can be used to communicate to a Fd process. However, there use is reserved to expert users and a regular user does not need to use them.

5.1. Cm control messages which could be sent to a Fd process

In addition to the Cfg messages, a Fd process will answer to the following messages

5.1.1. Message FdGetChannelsList (formally FdFrameChannels)

A process sending frames through Ethernet may build the list of channels it has seen since a given time (see the parameters of the FDOUT_CM_SERVER key). To request this list of channels through a Cm message you can send a Cm command with type "FdGetChannelsList" without any additional parameters. For instance to request a list of channels to FbmMoniUsers:

```
cm send -to FbmMoniUsers -type FdGetChannelsList –int gps
```

Remarks

- The parameter GPS is used only by dataSender to return the channel list for a given GPS time.
- The Fd process will answer to this request by the FdChannelsList message described below.
- This type of message is automatically sent by processes like dataDisplay or FdWrite.
- If the maximum number of Cm connection has been set to zero (first parameter of the FDOUT_CM_SERVER key), the process will not answer to the request.
- The FdGetChannelsList tool is available for regular user to get the channels list of a process, or as an example for developers.
- A developer could use the FdRequestChannelsList function to perform the request.

5.1.2. Message FdAddCmOuput (formally FdAddFrame)

To add a frame output, you just need to send a Cm message with type *FdAddFrameOutput*. For instance, to add to the server "FbmMoniUsers" an output named "display1" that will receive all channels:

```
cm send -to FbmMoniUsers
           -type FdAddCmOutput
           -text display1    // the Cm destination name
           -text "*"         // the tag for the channels list
           -int 5            // the output queue size
           -int 0            // the number of retry; 0 means for ever
```

Remarks:

- This message is equivalent to add a FDOUT_CM key in the configuration file
- This type of message is automatically sent by processes like dataDisplay or FdWrite.
- Cm output could only ba added in the limit of the maximum number of Cm outputs defined by the FDOUT_CM_SERVER key.
- The answer provided by the Fd process is the emission of the frame over Cm (see the FdFrame message below).
- A developer could use the FdRequestFrames function to perform the request.

5.1.3. Message FdRemoveCmOutput (formally FdRemoveFrame)

To remove a Cm output, you just need to send a Cm message with type *FdRemoveCmOutput*. For instance, to remove from the server "FbmMoniUsers" an output named "display1" :

```
cm send -to FbmMoniUsers -type FdRemoveCmOutput -text display1
```

Remarks:

- This type of message is automatically sent by processes like dataDisplay or FdWrite.
- A developer could use the FdRemoveFramesRequest function to perform the request.

5.1.4. Message FdAddFrameMergerSource

To add a frame source to a frame merger you need to send a Cm message with type *FdAddFrameMergerSource*. For instance, to add to the server "FrameMerger" the frame source”ImagingWE" :

```
cm send -to FrameMerger -type FdAddFrameMergerSource -text ImagingWE
```

Remarks:

- A message will be print in the FrameMerger log file.
- If this is a permanent change, don’t forget to update the frameMerger configuration.

5.1.5. Message FdRemoveFrameMergerSource

To remove a frame source from the input list of a frame merger you need to send a Cm message with type *FdRemoveFrameMergerSource*. For instance, to remove from the server "FrameMerger" the frame source ”ImagingWE" :

```
cm send -to FrameMerger -type FdRemoveFrameMergerSource -text ImagingWE
```

Remarks:

- A message will be print in the FrameMerger log file.
- If this is a permanent change, don’t forget to update the FrameMerger configuration.

5.1.6. Message FdSetNextStop

To tell an Fd process to stop at a given time, you can send a Cm message with type *FdSetNextStop*. The message has a single integer argument: a modulo time defining the next stop. For instance,

```
cm send -to FrameMerger -type FdSetNextStop -int 100
```

Will make the process named FrameMerger to stop when the GPS time of its output frame reaches a number modulo 100.

Remarks:

- If the argument is larger than the current GPS time, it is equivalent to request a stop at this specific time.
- If the argument is zero, it will reset a pending request to stop and the process will not stop.
- If the argument is negative, it will use the predefined value, either 1 or the value provided by the FDOUT_SET_NEXT_STOP key (see 4.5.20).

5.2. Cm messages returned by a Fd process

5.2.1. Message FdChannelsList

This is the answer to the FdGetChannelsList Cm request. The content of the message is the following:

- an integer to give the GPS time of the list,
- the “ADC” string,
- an integer to give the number of FrAdcData channels,
- a text to give the list of FrAdcData channels with their sampling rate,
- the “PROC”,
- an integer to give the number of FrProcData channels,
- a text to give the list of FrProcData channels with their sampling rate,
- the “SER” string,
- an integer to give the number of SerData channels (name of the FrSerData structure + name of the channel itself),
- a text to give the list of SerData channels with their sampling rate,
- the “SIM” string,
- an integer to give the number of FrSimData channels,
- a text to give the list of FrSimData channels with their sampling rate,

5.2.2. Message FdFrame

This is the message used to send frames vi Cm to another process, which could also be a Fd process. The format used to send frames on the Ethernet network is the same as the one used to write frames on disk plus a few header words extracted from the frame header. The type of the Cm message is "FdFrame" and its content is:

- A text for the experiment name (frame->name)
- An int for the Run number (frame->run)
- An int for the Frame number (frame->frame)
- An int for the GPS starting time in sec. (frame->GTimeS)
- An int for the GPS starting time, nsec. (frame->GTimeN)
- An int for the Quality/Trigger information (TBD)
- An int for the size of the frame
- A char* for the frame itself written with Frame library function FrameWriteToBuf.
- A double for the frame duration (frame->dt). If more than one frame is send in the same message, this is to total length of the set of frame.

It is recommended to use the Frame Distribution library API's to encapsulate a frame into a Cm message.

6. Package installation

This section describes the installation procedure for the Fd library. Since it is a Virgo core library, a summary of the Virgo software framework and management is given. This installation procedure has been checked on Scientific Linux v6.

6.1.1. Virgo Software Package organization

Most Virgo software is organized in packages which are managed using CMT and have a standard hierarchical structure which allow to keep multiple version of the same package as well as multiple binaries architecture on the same file server:

<package name>/ <v#r#p#>/cmt	for the compilation tools
/ doc	for the documentation
/src	for the sources and headers
/Linux-i686-SL5	for the binaries for this architecture
/Linux-x86_64 ...	for another platform

CMT handles the dependencies and allows consistent recompilation of portions of the software which depend on packages being modified.

6.1.2. Installing CMT

The CMT installation is described in the install page of the CMT web site (http://www.cmtsite.net). In practice you have to do the following actions:

- Create the CMTPATH variable (you must include this line in your .cshrc file). This is the directory where all the standard Virgo packages will be stored, create the directory and go into it:
  ```
  setenv CMTPATH /somewhere/virgoApp/
  mkdir $CMTPATH
  cd $CMTPATH
  ```
- Download the CMT source kit. This procedure has been tested with version v1r24:
  ```
  wget http://www.cmtsite.net/v1r24/CMTv1r24.tar.gz
  ```
  Remark: in case of problem with wget you can download the tar file from the download section of http://www.cmtsite.net.
- Extract the CMT source and prepare the installation:
  ```
  tar xvfz CMTv1r24.tar.gz
  cd CMT/v1r24/mgr
  ./INSTALL
  ```
- Setup the CMT environment variable:
  ```
  source $CMTPATH/CMT/v1r24/mgr/setup.csh
  ```
  It is recommended that you include the command in your .cshrc file to be able to reuse CMT at your next login.
- Build CMT:
  ```
  make
  ```
- At this stage CMT is installed. This could be check by running the command:
  `cmt show version` which should return the version number: v1r24

6.1.3. Installing PackageManagement

This is the package which manages for the Virgo packages installation

- First create the folder to store the package and download the package
  ```
  mkdir $CMTPATH/PackageManagement
  cd    $CMTPATH/PackageManagement
  setenv SVNROOT https://svn.ego-gw.it/svn/advsw/
  svn co --username=lcgt $SVNROOT/PackageManagement/tags/v2r9
  ```
- Then build the package using CMT
  ```
  cd v2r9/cmt
  cmt config
  make
  ```
- Finally initialize the package. It is recommended that you include the command in your .cshrc file to be able to reuse CMT at your next login.
  ```
  setenv PM_INSTALLATION_DIR $CMTPATH
  source $CMTPATH/PackageManagement/v2r9/cmt/setup.csh
  ```

```
setenv UNAME $CMTBIN
```

6.1.4. Installing the "Fd" package

The Fd package could then be installed using the following command:

```
cd $CMTPATH
Pm install -y -i root Fd v8r00
```

This command is slow (many seconds before the first output) because it is checking the SVN repository. After downloading the needed packages, the installation will begin. With the –i root, it will ignore the root package when installing Fd, which avoid the long installation of the CERN package Root.

6.1.5. Configure and testing Fd/Cm

Create a file with the CascinaLV domain information (only the first time):

```
setenv VIRGODATA /somewhere
mkdir $VIRGODATA
mkdir $VIRGODATA/Cm/
mkdir $VIRGODATA/Cm/mgr
echo CascinaLV cmns.virgo.infn.it 30000 30001 899 $VIRGODATA/Cm > $VIRGODATA/Cm/mgr/CmDomains
```

Define a few variables. The three next lines should be in your .cshrc file for future use

```
setenv CMROOT $CMTPATH/Cm/v8r9/
```

(Remarque: replace v8r9 by the version number of Cm you installed)

```
source $CMROOT/cmt/setup.csh
setenv CMDOMAIN CascinaLV
```

Then you need to open the TCPIP port 30000 to 30100 from/to your computer to the Cascina domain. The base IP address and netmask for the virgo.infn.it domain are:

```
193.205.72.0 netmask 255.255.255.0
193.205.73.0 netmask 255.255.255.0
193.205.74.0 netmask 255.255.255.0
193.205.75.0 netmask 255.255.255.0
```

Finally test it with the command:

```
cm names
```

which should return a list of process running at Cascina.
At this point you should be able to send and receive frames using the Fd library.

6.1.6. Summary of the commands which are useful to keep in your .cshrc

```
setenv CMTPATH /somewhere/virgoApp/
source $CMTPATH/CMT/v1r24/mgr/setup.csh
setenv SVNROOT https://svn.ego-gw.it/svn/advsw/
setenv PM_INSTALLATION_DIR $CMTPATH
source $CMTPATH/PackageManagement/v2r2/cmt/setup.csh
setenv UNAME $CMTBIN
setenv VIRGODATA /somewhere
setenv CMROOT $CMTPATH/Cm/v8r9/
source $CMROOT/cmt/setup.csh
setenv CMDOMAIN CascinaLV
```

6.1.7. Installing the code without Pm or debugging cmt make problems

In case of problem of installation with the Pm install scripts, it is possible to do the installation in steps. Each package could be extract from the Virgo SVN server buy doing commands like:
"svn co $SVNROOT/Fr/tags/v8r22 "
Then go it the cmt directory, run the command "cmt config" and finally "make".
It is possible to see more details by either doing "make VERBOSE=1" or setting the environment variable "setenv VERBOSE 1" (use unsetenv VERBOSE to remove it).

7. History

7.1. v8r00 (October 2nd, 2014)

Major rewrite of the library and associated applications. The main changes are:

- Provide a full control of the sequence of operations.
- Remove FD_SHM keys; this is replaced by the use of the FDIN_DIR keys reading /dev/shm.
- Upgrade the FD_DIR_CLEAN keys, produce ffl at the same time if requested.
- Do a strong check of the unused words in the configuration.
- Provide better support for program state.
- Add multi-thread option for faster compression.
- Add the tagging when reading from directory.
- The Cm output could now work in asynchronous mode (post) in case of multiple outputs. This does not bloc anymore to other receiving process.
- Data sender is put aside temporarily.
- Rename the FdIOIni and FdIOParse functions (keep the old ones for backward compatibility)
- Rename the functions to put/get frames from Cm messages (FdPutFrameInMessage, FdGetFrameFromMessage).
- Update the names and parameters of the Cm messages to request frames. This make this version incompatible with the previous
- Reset the history log. To see the older history, download the Fd/v7r06 version which has the latest history for version 7.

7.2. v8r00p1 (October 8th, 2014)

Move to Cfg v8r00p5 instead of v8r00p4.

7.3. v8r01 (October 18th, 2014)

- Move to Cfg v8r00p5 instead of v8r00p4.
- Increase the size of the Cfg mailbox
- Improve the FD_NO_CONFIG_CHECK key: in case of unused key, the parsing of the configuration does not start.
- Fix bugs in the software injection code.
- FdFilter.c: Fix a bug when computing the reduced frequency of the Butterworth filters: due to rounding effect observed on SL4, the output frequency was not always the requested frequency.
- FDOUT_FILE: fix a bug to extract the file name prefix if multiple directories are used.
- Fix a bug in trend frame when the slope is negative: min and max were swapped.
- Fix unused variables in various file to remove compilation warnings.

7.4. v8r02 (November 6th, 2014)

- Move to Cfg v8r01.
- Fix bugs in the software injection code.
- Fix a bug with the FDIN/OUT_FRAME_DURATION keys which create a segmentation fault when the frame has already the right size.

7.5. v8r03 (November 18th, 2014)

- Move to Cfg v8r02.
- Remove fake closing output file error message if the writing stop at the end of a file change (i.e. file reaching the requested length)
- Add the function FdResizeAdd.
- Fix a memory leak occurring when reducing the frame duration

7.6. v8r04 (December 7th, 2014)

- Use Cm/v8r12 and Fr/v8r23

- FDOUT_FILE : add the creation the folder in /dev/shm if they are not present
- FDOUT_CM add a parameter to tell to compute the checksums or not;
- Add the keys FDIN/OUT_REJECT_EVENTS
- Add a consistency check on the buffer size when receiving frames from Cm
- Fix bugs in ffl preparation with the FDOUT_CLEAN_DIR_FFL key
- Bug fix when receiving frames from Cm to avoid a segmentation fault at the first received frames when the number of possible sources was set to zero (meaning no check).

7.7. v8r04p1 (December 13th, 2014)

- Move to Cm /v8r12p1 which add protections for incomplete incoming messages.

7.8. v8r05 (January 7th, 2015)

- Move to Cm /v8r12p2, Cfg/v8r03, Frv/v4r22p2.
- FDIN_DIR: change the logic when we are late reading the input buffer: now jump to the most recent file.
- FDIN_TAG: if the input tag is "*" read the full frames directly instead of reading all the channels.
- FDIN_TAG: fix it since it was not selecting events.

7.9. v8r06 (January 27th, 2015)

- Move to Cfg/v8r03p1 and therefore Cm /v8r12p3.
- Fix various compilation warnings in printf/CfgMsg…
- FdCompress: add some sleep time to not block the start of thread at initialization.
- FDIN_DIR : if the directory is empty ; keep scanning it until a .gwf file is provided instead of stopping on a FATAL error.
- FDIN_DIR: Change the severity of a message from ERROR to INFO and request debug level > 0 to print it

7.10. v8r07 (February 28th, 2015)

- FDIN_FRAME_MERGE: set the status to active if at least one of the expect source is missing
- FDIN_DIR:
  - Do not call the other input actions if there are no new frames. This is to avoid incomplete frame when the FDIN_FRAME_DURATION is used to extend the frame length.
  - Put the error message: 'run a full directory scan' under the debug level flag control
- FDIN_S_INJECTION_FILE :
  - Reduce the number of error messages if the injection channel is missing
  - Change from ERROR to FATAL when the h(t) channel to be injected do not have the right type.
  - Bug fix: the requested start time was always reset to the beginning of the file
- Add the key FDIN_COMPRESSION and set the default compression flag to -1 instead of 0 when reading from files
- FDOUT_CLEAN_DIR_FFL: add the overlap parameter
  - Update trendFrameBuilder.cfg to show the use of the new FDOUT_CLEAN_DIR_FFL parameter
- FdOFileProcess: fix an invalid read access when closing an output file
- FDXX_BUT_FILTER: change the frequency suffix generation in the channel name to keep only one frequency suffix.
- Move to Frv/v4r23 and Fr/v8r24

7.11. v8r08 (March 5th, 2015)

- FDIN/OUT_STAT: Add a protection against ADC channels without associated vectors like it could be the case if the sampling rate is changed and the frame duration is extended during such a change.

7.12. v8r09 (April 12th, 2015)

- FDIN/OUT_BUT_FILTER: check the consistency between sampling rate, vector and frame size and set the action state to active instead of golden in case of error with a channel, reset the filter if the vector size or sampling rate change and improve the reporting of errors and filter register reset.

- Software injections:
    - FDIN_S_INJECTION_CHANNEL: add a third parameter to copy to the FrSimEvent structures.
    - Apply the scaling factor on the FrProcData recorded injection channel
    - Bug fix for software injections: if the FDIN_S_INJECTION_EVT_NAME was not used, the injections were not made due to a scale factor set to zero.
    - FDIN_S_INJECTION_RESCHEDULE: add a protection if the key FDIN_S_INJECTION_EVT_NAME is missing
- Improve the text of the error message in case of unused word
- Move to Frv/v4r24 and therefore Fr/v8r25

7.13. v8r10 (April 27th, 2015)

- FD_TREND: propagate FrSerData units to the FrAdcData vector and not just to the FrAdcData structure to allow a correct display of the units.
- FDIN/OUT_BUT_FILTER:
    - Avoid the improper reset of the filters error message if the input is just zero.
    - In case of multiple inputs for the same output channel: selected the faster channel and improve the error reporting.
- FDOUT_FILE: Add protection when closing file to avoid wrong error message if output files are written with single frame.
- Add the Cm handler FdSetNextStop to stop a process at a given time.
- FDIN_FRAME_MERGER: put the latency information in the same FrSerData structure as the other parameters of the FdIOServer process; Now they all have the same prefix: V1:Daq, unless it is changed with the FD_CH_PREFIX key.
- Move to Frv/v4r25

7.14. v8r11 (May 17th, 2015)

- Add the FDOUT_SET_NEXT_STOP key.
- FDOUT_CM_SERVER:
    - Add the default queue size as third parameter to the FDOUT_CM_SERVER process.
    - bug fix: the cm outputs were not removed from the output list if only 0 try was requested
- Move to Frv/v4r25p1 and therefore Fr/v8r26 to fix a problem when resizing frame with very slow channel (with a period larger than the frame duration).

7.15. v8r12 (May 26th, 2015)

- Improve the user info message
- Move to Frv/v4r26 to avoid fake error messages about filter reset due to sub-normal numbers.

7.16. v8r13 (August 18th, 2015)

- Add the code for the FDOUT_CONVERT_SERDATA key
- Upate a test program to use the FDOUT_SELECT_CHANNELS key
- Add the FDIN/FDOUT_SELECT_CHANNELS keys
- Bug fix : the FDOUT_CM keys were producing a segmentation fault due to the access of an uninitialized internal variable

7.17. v8r14 (January 3rd, 2016)

- Move to Frv/v4r27
- Bug fix: avoid to count units as duplicated channels
- Reduce the amount of log messages when maintaining the list of channels
- Add the FDIN_RANGE_GATING key.

7.18. v8r15 (January 16, 2016)

- FdStat: add nData and rate info
- Bug fix for the channel list: if the sampling rate of a channel change, update it.