The Frame Distribution (Fd)
User's guide

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

Version: v8r42
March 13, 2022

TABLE OF CONTENTS

1. INTRODUCTION 6

2. BASIC USE OF THE FD LIBRARY 7

2.1. INTRODUCTION 7
2.2. DESCRIPTION OF THE BASIC FDIO FUNCTIONS 7
2.2.1. FdIONew 7
2.2.2. FdIOParseAndIni 7
2.2.3. FdIOGetFrame 7
2.2.4. FdIOPutFrame 8
2.2.5. FdSetUserState 8
2.2.6. FdStatBuild 8
2.3. EXAMPLE OF APPLICATION: FDMONI 8
2.4. INFORMATION FOR ADVANCED DEVELOPERS TO SEND/RECEIVE FRAMES OVER CM 8
2.4.1. Function to put a frame in a Cm messags: FdPutFrameInMessage 8
2.4.2. Function to get a frame from a Cm message: FdGetFrameFromMessage 9
2.4.3. Function to request frames via Cm: FdRequestFrames 9
2.4.4. Function to remove a frame request: FdRemoveFramesRequest 9
2.4.5. Function to request the list of channel: FdRequestListOfChannels 9

3. BASIC PROGRAMS AND TOOLS 10

3.1. FDIOSERVER 10
3.2. EXAMPLES OF FDIOSERVER CONFIGURATIONS 10
3.2.1. simChannels.cfg 10
3.2.2. copyFile.cfg 10
3.2.3. dupChannels.cfg 10
3.2.4. trendFrameBuilder.cfg 11
3.2.5. demodulation.cfg 11
3.2.6. writeDir.cfg + readDirDy.cfg 11
3.2.7. cmSend.cfg + cmReceive1.cfg, cmReceive2.cfg and cmReceive3.cfg 11
3.2.8. frameMerger.cfg + toFrameMerger_1.cfg and toFrameMerger_2.cfg 11
3.2.9. playback.cfg 11
3.2.10. softwareInjections.cfg 11
3.2.11. noConfigCheck.cfg: 12
3.3. FDGETCHANNELSLIST 12
3.4. FDSEND 12
3.5. FDSTAT 12
3.6. FDWRITE 12

4. CONFIGURATION PARAMETERS 13

4.1. CONFIGURATION ACCESS 13
4.2. KEYWORDS TO CONTROL GLOBAL PARAMETERS 13
4.2.1. FD_CH_PREFIX 13
4.2.2. FD_DEBUGLVL 13
4.2.3. FD_NO_CONFIG_CHECK 13
4.2.4. FDIN_NO_FR_MSG_TIMEOUT 13
4.2.5. FDIN_CM_TIMEOUT 13
4.2.6. FDOUT_CM_TIMEOUT 13
4.2.7. FD_REPORT_FR_ERRORS 13
4.3. KEYWORDS TO DEFINE AND CONTROL THE SOURCE OF INPUT FRAMES 13
4.3.1. FDIN_ADD_FROM_FILE 14
4.3.2. FDIN_ ADD_NO_INJECTION_CHANNEL 14
4.3.3. FDIN_CM 14
4.3.4. FDIN_CM_CONNECT 14
4.3.5. FDIN_DATA_SENDER 15
4.3.6. FDIN_DIR 15
4.3.7. FDIN_FILE 15

4.3.8. FDIN_FRAME_MERGER ... *15*
4.3.9. FDIN_MAX_SPEED ... *16*
4.3.10. FDIN_MERGE_FILES ... *16*
4.3.11. FDIN_MIN_LATENCY ... *16*
4.3.12. FDIN_NEW_FRAME ... *16*
4.3.13. FDIN_PLAYBACK ... *17*
4.3.14. FDIN_SLEEP ... *17*
4.3.15. FDIN_TAG ... *17*
4.4. KEYWORDS TO CONTROL THE INPUT FRAMES CONTENT ... 18
4.4.1. FDIN_ADD_PREFIX ... *18*
4.4.2. FDIN_ADD_SUFFIX ... *18*
4.4.3. FDIN_ADD_SURROGATE_CHANNEL ... *18*
4.4.4. FDIN_ADD_SURROGATE_DQ_CHANNEL ... *18*
4.4.5. FDIN_ADD_TEST_CHANNEL ... *19*
4.4.6. FDIN_BITOP ... *19*
4.4.7. FDIN_BUT_FILTER ... *19*
4.4.8. FDIN_COMBINE_CHANNELS ... *20*
4.4.9. FDIN_COMBINE_FLAGS ... *20*
4.4.10. FDIN_COMPRESSION ... *20*
4.4.11. FDIN_CONVERT_NTCT ... *20*
4.4.12. FDIN_CONVERT_SERDATA ... *21*
4.4.13. FDIN_DOWN_SAMPLING ... *21*
4.4.14. FDIN_DUPLICATED_CHANNELS_CHECK ... *21*
4.4.15. FDIN_DUPLICATED_CHANNELS_REMOVE ... *21*
4.4.16. FDIN_ FLAG_MIN_MAX ... *21*
4.4.17. FDIN_FFT_FILTER ... *22*
4.4.18. FDIN_FRAME_DURATION ... *22*
4.4.19. FDIN_FSHIFT ... *22*
4.4.20. FDIN_DEMOD ... *22*
4.4.21. FDIN_GET_BRMS ... *23*
4.4.22. FDIN_GET_LINE ... *23*
4.4.23. FDIN_GET_LINES ... *23*
4.4.24. FDIN_GET_MODULUS ... *24*
4.4.25. FDIN_GET_PHASE ... *24*
4.4.26. FDIN_HISTORY ... *24*
4.4.27. FDIN_NOMSG ... *24*
4.4.28. FDIN_RANGE_GATING_NEW ... *24*
4.4.29. FDIN_RECAST_CHANNELS ... *25*
4.4.30. FDIN_RECOLOR_CHANNELS ... *26*
4.4.31. FDIN_REJECT_EVENTS ... *26*
4.4.32. FDIN_RENAME_CHANNEL ... *26*
4.4.33. FDIN_RESCALE_CHANNEL ... *26*
4.4.34. FDIN_S_INJECTION_FILE ... *27*
4.4.35. FDIN_S_INJECTION_EVT_NAME ... *27*
4.4.36. FDIN_S_INJECTION_FLAG ... *27*
4.4.37. FDIN_S_INJECTION_RESCHEDULE ... *27*
4.4.38. FDIN_S_INJECTION_RESCALE ... *27*
4.4.39. FDIN_S_INJECTION_CHANNEL ... *28*
4.4.40. FDIN_SEGMENTS ... *28*
4.4.41. FDIN_SELECT_CHANNELS ... *28*
4.4.42. FDIN_SELECT_FRAMES ... *28*
4.4.43. FDIN_SET_PROC_TYPE ... *28*
4.4.44. FDIN_STAT ... *28*
4.4.45. FDIN_TREND_OF_TREND ... *29*
4.4.46. FDIN_VETOS ... *29*
4.5. KEYWORDS TO CONTROL THE FRAME OUTPUT(S) ... 30
4.5.1. FDOUT_ADD_PREFIX ... *30*
4.5.2. FDOUT_ADD_SUFFIX ... *30*
4.5.3. FDOUT_BUT_FILTER ... *30*
4.5.4. FDOUT_BITOP ... *30*

4.5.5. FDOUT_DOWN_SAMPLING ... 30
4.5.6. FDOUT_DUPLICATED_CHANNELS_CHECK ... 30
4.5.7. FDOUT_DUPLICATED_CHANNELS_REMOVE ... 30
4.5.8. FDOUT_CLEAN_DIR ... 30
4.5.9. FDOUT_CLEAN_DIR_FFL ... 30
4.5.10. FDOUT_CM ... 31
4.5.11. FDOUT_CM_SERVER ... 31
4.5.12. FDOUT_COMBINE_CHANNELS ... 32
4.5.13. FDOUT_COMBINE_FLAGS ... 32
4.5.14. FDOUT_COMPRESSION ... 32
4.5.15. FDOUT_CONSUMER ... 32
4.5.16. FDOUT_CONVERT_NTCT ... 32
4.5.17. FDOUT_CONVERT_SERDATA ... 32
4.5.18. FDOUT_DELAY ... 32
4.5.19. FDOUT_FFT_FILTER ... 32
4.5.20. FDOUT_FILE ... 32
4.5.21. FDOUT_FILE_CHECKSUM ... 33
4.5.22. FDOUT_FREQUENCY_CUT ... 33
4.5.23. FDOUT_FRAME_DURATION ... 33
4.5.24. FDOUT_GET_BRMS ... 33
4.5.25. FDOUT_GET_LINE ... 33
4.5.26. FDOUT_GET_LINES ... 33
4.5.27. FDOUT_HISTORY ... 33
4.5.28. FDOUT_RECAST_CHANNELS ... 33
4.5.29. FDOUT_RECOLOR_CHANNELS ... 33
4.5.30. FDOUT_RENAME_CHANNEL ... 33
4.5.31. FDOUT_REJECT_EVENTS ... 33
4.5.32. FDOUT_RESCALE_CHANNEL ... 33
4.5.33. FDOUT_SEGMENTS ... 33
4.5.34. FDOUT_SELECT_CHANNELS ... 33
4.5.35. FDOUT_SELECT_FRAMES ... 33
4.5.36. FDOUT_SET_NEXT_STOP ... 34
4.5.37. FDOUT_SET_PROC_TYPE ... 34
4.5.38. FDOUT_SHM ... 34
4.5.39. FDOUT_SKIP_FRAMES_WITHOUT_EVENT ... 34
4.5.40. FDOUT_STAT ... 34
4.5.41. FDOUT_THREAD ... 34
4.5.42. FDOUT_TO_ACL ... 34
4.5.43. FDOUT_TREND_FRAME ... 35
4.5.44. FDOUT_TREND_OF_TREND ... 35
4.5.45. FDOUT_VETOS ... 35

5. CM MESSAGES AVAILABLE ... 36

5.1. CM CONTROL MESSAGES WHICH COULD BE SENT TO A FD PROCESS ... 36
5.1.1. Message FdGetChannelsList (formally FdFrameChannels) ... 36
5.1.2. Message FdAddCmOuput (formally FdAddFrame) ... 36
5.1.3. Message FdRemoveCmOutput (formally FdRemoveFrame) ... 36
5.1.4. Message FdAddFrameMergerSource ... 37
5.1.5. Message FdRemoveFrameMergerSource ... 37
5.1.6. Message FdSetNextStop ... 37
5.2. CM MESSAGES RETURNED BY A FD PROCESS ... 37
5.2.1. Message FdChannelsList ... 37
5.2.2. Message FdFrame ... 38

6. PACKAGE INSTALLATION ... 39

6.1.1. Virgo Software Package organization ... 39
6.1.2. Installing CMT ... 39
6.1.3. Installing PackageManagement ... 39
6.1.4. Installing the "Fd" package ... 40
6.1.5. Configure and testing Fd/Cm ... 40

6.1.6. Summary of the commands which are useful to keep in your .cshrc 40
6.1.7. Installing the code without Pm or debugging cmt make problems 40

7. HISTORY 42

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 3.1, 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 built 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. FdSetUserState

If the user wants to update the program state, including the state information from FdIO, he could use the function:

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

- A call with the state value less or 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 the state equal to CfgServerActive, CfgServerGolden, or CfgServerError will issue a call the CfgReachState with a state value that will take the worse value of this input value and all the FdIO actions.

If the user set fdIO->autoState = CfgTrue; then each call to FdIOGetFrame and FdIOPutFrame will update the program state (CfgServerActive, CfgServerGolden, or CfgServerError), accordingly to the possible problem of the input/output frames (this was the default up to version v8r23).

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 is 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 asks to the 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 examples 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_ADD_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 forever.
- 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 with 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 built 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.

4.2.7. FD_REPORT_FR_ERRORS

This key has no parameters.
If this key is present, the FrameL errors will be reported as Cfg error message and in the log file.

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

Only one source of frame could be used. If your 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_ADD_FROM_FILE

This key is designed to add data (channels or events) from a frame file to the main input stream:

- Parameter 1 (string): The name of the file to be added. The frame duration should be the same as the one of the main stream.
- A tag to select the channel and events from this file. Use "*" to select everything
- An optional time offset to be applied on the event time to do time shift coincidences

For instance, the configuration

FDIN_FILE file1.ffl 1233338100 100
FDIN_ADD_FROM_FILE file2.ffl "*"
FDIN_ADD_FROM_FILE file3.ffl "*"

Will add all data from file2 and file3. Notice that if there are gaps in the file used by the FDIN_FILE, none of the files will be read at that time. To avoid this possible issue, it is possible to have the root frame generated on the fly using this kind of configuration

FDIN_NEW_FRAME MbCoinc 1 1233338100 100

FDIN_ADD_FROM_FILE file1.ffl "*"
FDIN_ADD_FROM_FILE file2.ffl "*"
FDIN_ADD_FROM_FILE file3.ffl "*"

4.3.2. FDIN_ ADD_NO_INJECTION_CHANNEL

This key is designed to create a channel to monitor if injections are performed or not. It uses the FrSimEvent information from an auxiliary file.

- Parameter 1 (string): name of the no_injection channel to be created. The channel is of type “char”. It is set to 1 when there are no injections, and 0 during injection time.
- Parameter 2 (int): sampling rate (Hz) of the no_injection channel
- Parameter 3 (real): margin before: this is the time (s.) before the FrSimEvent time set as injection time.
- Parameter 4 (real): margin after: this is the time (s.) after the FrSimEvent time set as injection time.
- Parameter 5 (string): name of the file where the FrSimEvent are searched.
- Parameter 6 (string): tag to select the FrSimEvent

4.3.3. 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. Must be larger than zero. 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 tries to send frames, these frames will be rejected.

4.3.4. 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 stops, a message is sent 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).

4.3.5. FDIN_DATA_SENDER

This keyword let you connect to a "dataSender", a remote source of frame sending them 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.

4.3.6. 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 starts 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.

4.3.7. 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). Remark: more complex time selection could be made using the SEGMENT tool of the ffl.

The structure checksums are checked during the read process. In case of read error (checksum or other type), the process abort.

4.3.8. FDIN_FRAME_MERGER

- Parameter 1 (int) number of frames 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 frames: FrameDepth (first parameter of the key).
- Frame parts arriving after 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 another 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.

4.3.9. 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.

4.3.10. FDIN_MERGE_FILES

This keyword let you merge frames from multiple input files overlapping in time.
This key could not be used with another key defining the input frame like FDIN_NEW_FRAME

- Parameter 1 (string): The set of input files like "file1.gwf file2.gwf" were file1 and file2 could have overlapping time period. After opening all the listed files, the processing finds the first one, and if there are frames with the same GPS time from different files, they are merged. The frame duration must be the same, or the program stop with an error. It is possible to use "*" is a file name the select multiple files: "file*.gwf" will merge all files matching this search.
- Parameter 2 (integer): Start GPS time (or beginning of the file if this parameter is set to zero or missing).
- Parameter 3 (integer): Number of seconds of data to read (or until the end of the file if this parameter is missing)

4.3.11. 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

4.3.12. 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.
    - The must be a multiple of the frame length.
    - 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 forever.
- Parameter 5 (integer): Data Quality flag put in the "dataQuality" word of the frame header.

4.3.13. 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 the data
- Parameter 5 (integer): number of times 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:
  $Ti = Tr - L*(int)((Tr-GPSstar)/ lenReplay)$
- A 1 Hz channel named V1:FdPlayback_original_GPS is created. As its name says, it contains the original GPS time of the frame replayed.
- 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 stops 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.

4.3.14. 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.

4.3.15. 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 or events on the input frames.

4.4. Keywords to control the input frames content

4.4.1. FDIN_ADD_PREFIX

- Parameter1 (string): Prefix to be added to the name of each channel (like: "V1:").
- Parameter2 (string): Tag to select the channel(s) on which the prefix is added (like "*AC*"). If this parameter is missing, the prefix is added to all channel

With this key the prefix defined as 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.

4.4.2. FDIN_ADD_SUFFIX

- Parameter1 (string): Suffix to be added to the name of each channel (like: "_DS").
- Parameter2 (string): Tag to select the channel(s) on which the suffix is added (like "*AC*"). If this parameter is missing, the suffix is added to all channel

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

4.4.3. 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). If the spectral density is given by a vector, two types of vectors 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 separated by a space. Additional characters are allowed on each line after these two values.
    - The spacing of the frequencies and the number of lines is arbitrary, as long as the values are given in increasing order.
- 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.

4.4.4. FDIN_ADD_SURROGATE_DQ_CHANNEL

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

4.4.5. 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.

4.4.6. FDIN_BITOP

This performs bitwise operation on flags (channels of type int or unsigned int).

- Parameter 1 (string): Name of the output channel, could be the input channel.
- Parameter 2 (string): Name of the input channel
- Parameter 3 (string): Operation: one of the following: "NOT","AND","OR" ,"XOR","<<",">>" or "MASK".
- Parameter 4 (hex): Parameters or mask applied. In the case of “NOT” a mask is also apply on the result of the NOT operation (output = (~input) & mask).
- Parameter 5 (int): Output frequency. If this optional parameter is present and if the requested frequency is larger than the input frequency, the output channel is produced at this new frequency, repeating the output value to match the new output frequency. If this parameter is missing, the output sampling rate is the same as the one of the input channels.

4.4.7. 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, a new suffix is also added since version v8r25 (before it was replacing by the new one). 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. The sign of this parameter defines the filter as a low pass (for positive frequency) of high pass (for negative value). In the case of a high pass filter, the output channel has the suffix HP added.
- 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 channels produced by the previous FDIN key.

4.4.8. 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 channel 1.
- 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 channels are of type float (4 bytes).

4.4.9. FDIN_COMBINE_FLAGS

This key let you combine two flag channels (i.e. type int or unsigned int), creating a new one. The new channel is

output = channel1 >> shift1 + channel2 >> shift2

with

- Parameter 1 (string): name of the output channel. It could be one of the input channels.
- Parameter 2 string): channel1 name.
- Parameter 3 (double): shift1, if positive, bits are shifted to the left, if negative, to the right.
- Parameter 4 string): channel2 name.
- Parameter 5 (double): shift2, if positive, bits are shifted to the left, if negative, to the right.

Remarks:

- The output channel is created with the same type and sampling rate as channel1.
- Channel1 could be faster than channel2, but their relative rates must be an integer.

4.4.10. 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 (uncompressed 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.

4.4.11. FDIN_CONVERT_NTCT

This will do the conversion from Volts to degrees Centigrade, assuming the channel is a Negative Temperature Coefficient Thermistor (NTCT) temperature probe. The parameters are:

- Parameter 1 (string): name of the output channel (temperature) that is created.
- Parameter 2 (int): sampling rate of the output channel. Should be less than the input rate. It is rounded to an integer fraction of the input rate.
- Parameter 3 (string): name of the input channel. This should be the voltage of the NTCT.
- Parameter 4 (double): ADC resistance (ADC_R) in Ohms.
- Parameter 5 (double): NTCT resistance at T_{ref} in Ohms. (R_{ref})
- Parameter 6 (double): NTCT voltage in Volts (NTCT_V).
- Parameter 7 (double): beta coefficient.

- Parameter 8 (double): T_{ref} value.
- Parameter 9 (string): (optional) If present the NTCT resistance value is stored with this name.

First the resistance of the NTCT is computed as:

R= ADC_R * (NTCT_V - volts) / volts

Where volts is the value of the input channel

Then $T = 1./(\log(R/R_{ref})/beta + 1./T_{ref}) - 273.15$

4.4.12. FDIN_CONVERT_SERDATA

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

4.4.13. FDIN_DOWN_SAMPLING

With this keyword the 2d or 3d channels defined by a tag are down sampled to a new period. This is useful to reduce for instance the rate of the images or the sample channels.

- Parameter 1 (string): Tag defining the channel on which the down sampling is applied.
- Parameter 2 (integer): New period (in seconds) for the channel(s). This new period must be at least as long as the frame duration, meaning that we could have only one image (or zero) per frame.

4.4.14. 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 channels 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 use the FDIN_DUPLICATED_CHANNELS_REMOVE key.

4.4.15. 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. If this parameter is set to zero or has a negative value, duplicated channels are removed silently, without updating the program state and user info.

Remarks:

- The number of duplicated channels 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, and if reporting is requested (parameter1 > 0) the status of this Fd object is changed from “Golden” to “Active”

4.4.16. FDIN_ FLAG_MIN_MAX

This key defines a flag channel. This is an integer channel (unsigned 4 bytes int) whose value is:

- 0 if the input channel values are within a given range,
- 1 when the output channel is outside the given range,
- 2 if the input channel is missing.

The parameters are:

- Parameter 1 (string): name of the created flag channel.
- Parameter 2 (int) sample rate (Hz) of the flag channel. It should be at least the frame rate.
- Parameter 3 (string): name of the channel to monitor.
- Parameter 4 (double) minimal value of the input channel to keep the flag set to zero.
- Parameter 5 (double) minimal value of the input channel to keep the flag set to zero.

- Parameter 6 (double) padding: time (seconds) used to keep the flag set to one after the input values are back to normal.

4.4.17. 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.

4.4.18. 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.
- Parameter 2(float): firstGPS: This optional parameter defines the GPS time of the first frame to be produced. There are three different cases:
    - firstGPS is zero, negative or not provided (default case): the frame production starts right away. The first frame might be incomplete when extending the frame duration.
    - firstGPS = 1: The data are ignored until the start of a new frame. This is useful to simplify the production of long trend data frames.
    - firstGPS > 1: The data are ignored until GPS = firstGPS.

Remark: when extending the frame duration, if not enough frames are provided to make a full frame at the end of the processing (like at the end of an input file), the incomplete frame is lost. However, in the case of the FDOUT_FRAME_DURATION, the incomplete frame is returned at the end of the processing.

4.4.19. 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 dependent.

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.

4.4.20. 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.

4.4.21. FDIN_GET_BRMS

This object computes the band RMS for a channel (in the frequency domain).

- Parameter 1 (string): Name of the channel to be processed (FrAdcData or FrProcData).
- Parameter 2 (real): Period for the computation. A new FFT is performed with this period, without averaging. The period should be equal to the frame duration or an integer fraction of it. It could not be longer than the frame duration.
- Parameter 3 (real): starting frequency in Hz.
- Parameter 4 (real): end frequency in Hz.

Exemple :

FDIN_GET_BRMS V1:SIB2_B2_PD2_sample 10 1e5 5e5

Will compute every 10 seconds the BRMS of the channel V1:SIB2_B2_PD2 in the 100 kHz to 500 kHz. The name of the output channel will be: V1:SIB2_B2_PD2_sample_BRMS_100_500kHz. If the end frequency is lower than 100kHz, the name uses the frequency in Hz instead of kHz.

Remarque: the FDIN_GET_BRMS, FDIN_GET_LINE and FDIN_GET_LINES key analyzing the same channel should be placed next to each other to share the same FFT and optimize the computation.

4.4.22. FDIN_GET_LINE

This object computes the amplitude and phase of a frequency line. (in the frequency domain).

- Parameter 1 (string): Name of the channel to be processed (FrAdcData or FrProcData).
- Parameter 2 (real): Period for the computation. A new FFT is performed with this period, without averaging. The period should be equal to the frame duration or an integer fraction of it. It could not be longer than the frame duration.
- Parameter 3 (real): frequency of the line in Hz.

Two output signals are produced:

InputName_frequencyHz_mag for the amplitude
InputName_frequencyHz_phi for the phase.

Remarque: the FDIN_GET_BRMS, FDIN_GET_LINE and FDIN_GET_LINES key analyzing the same channel should be placed next to each other to share the same FFT and optimize the computation.

4.4.23. FDIN_GET_LINES

This object computes searches for the loudest lines in a frequency band.

- Parameter 1 (string): Name of the channel to be processed (FrAdcData or FrProcData).
- Parameter 2 (real): Period for the computation. A new FFT is performed with this period, without averaging. The period should be equal to the frame duration or an integer fraction of it. It could not be longer than the frame duration.
- Parameter 3 (real): starting frequency in Hz.
- Parameter 4 (real): end frequency in Hz.
- Parameter 5 (int): number of lines searched

For each line searched, two output signals are produced:

InputName_lineNumber_freq for the frequency of the line
InputName_lineNumber for the amplitude of the line.

Remarque: the FDIN_GET_BRMS, FDIN_GET_LINE and FDIN_GET_LINES key analyzing the same channel should be placed next to each other to share the same FFT and optimize the computation.

4.4.24. 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);

4.4.25. 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 dependent.

The output signal is computed as:

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

4.4.26. FDIN_HISTORY

This object replaces the full history of the frame header by only one history record. This key has only one parameter: the name of the new history record. This object is useful to simplify the frame history after complex processing.

4.4.27. FDIN_NOMSG

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

4.4.28. FDIN_RANGE_GATING_NEW

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 (FrAdcData or FrProcData) 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 (real): starting frequency (Hz) to compute the range. Typical value: 10 Hz.
- Parameter 4 (int): sample rate (Hz) to compute the range. Typical value: 32 Hz
- Parameter 5 (real): range threshold or fraction of median range (like 0.6). When the range is below this value, h(t) is set to zero. If this parameter is set to a negative value, the range is computed but no gating is applied.
- Parameter 6 (real): FFT duration in seconds. Typical value 0.25 seconds.
- Parameter 7 (real): Transition time in seconds to go from 0 to 1 (or 1 to 0) for the gate function (Tukey window). Typical value: 0.25 seconds
- Parameter 8 (int) (optional). If this parameter is present and non-equal to zero, it is the number of seconds on which the median range is computed then parameter 5, is the fraction of this median range used to define the threshold. Typical value: 10 seconds. If this parameter is missing, parameter 5 is an absolute range value in Mpc.
- Parameter 9 (string) (optional). If this parameter is present, it is a channel name used as an additional external trigger for the gating when its value is not equal to 1. The sampling rate of this channel should be the same as the sampling rate used to compute the gating.

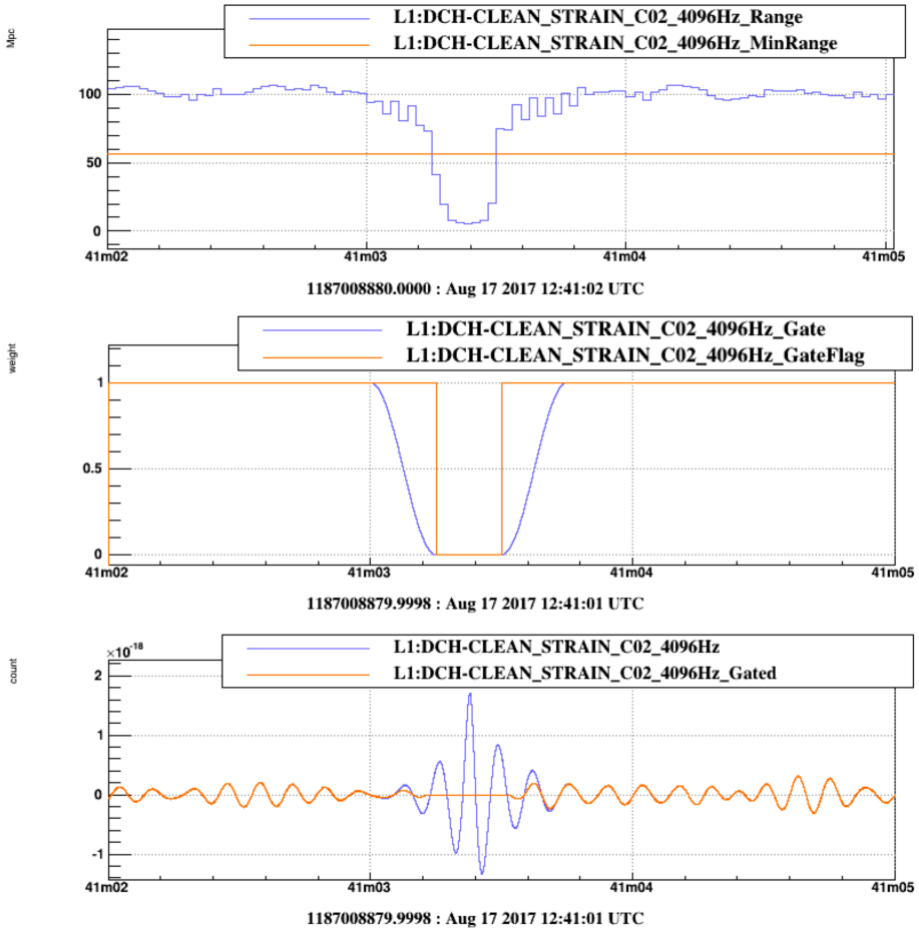
Remarks:

- This key introduces a latency of one frame in the pipeline.
- To smooth the range computation and remove the down fluctuation of the PSD when using only one FFT, the local PSD is taken as the maximum of the single PSD and the median PSD. This introduces a bias of

the range toward low range value. Therefore, a 1.2 scaling factor is applied on the range for a rough correction. This factor is not very accurate and depends on the various parameters used to compute the range.

- In case of frame discontinuity, the Tukey window is applied to smooth the transition when the h(t) channel appear/disappear.

Example:
The following figure is showing the time series for the inspiral range of L1 data near GW170817 (top plot), the weigth apply on the h(t) time serie and the gate flag (middle plot) and on the bottom plot the raw h(t) (blue curve) together wigth the gated h(t) (red curve). A loud glitch is producing a drop of the inspiral range which then trigger the gating.



4.4.29. 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 preserved (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.

4.4.30. FDIN_RECOLOR_CHANNELS

This object changes apply a user defined transfer function on one or more channel. An output FrProcData channel of type double, having the suffix "_recolored", is created for each processed channel.

- Parameter 1 (string): A tag which defines the channel or group of channels going to be recolored. 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 (string): the name of the transfer function file. Two types of vectors 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 transfer function amplitude separated by a space. Additional characters are allowed on each line after these two values.
    - The spacing of the frequencies and the number of lines is arbitrary, as long as the values are given in increasing order.

4.4.31. 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 which the rejection criteria will be applied. 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 rejection. It could be any event parameter or the event amplitude (use in this case "amplitude").
- Parameter 3 (double): min value used for the rejection.
- Parameter 4 (double): max value used for the rejection.
- Parameter 5 (int): down sampling factor. This is the reduction factor applied on the rejected 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 rejection criteria 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 rejection 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

4.4.32. 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

4.4.33. 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 vectors.

- 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)

4.4.34. 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 (or if this parameter is missing) means start at the file begin.
- Parameter 3 (integer): duration of the input file to read. Zero (or if this parameter is missing) means use the full file.

4.4.35. 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.
- Parameter 2 (string): new name of the FrSimEvent structures. If "." is used, or if this parameter is missing, the names are unchanged.
- Parameter 3 (integer): This is the size in seconds of the data chunk read to search for the start and stop of the injection channel. If this optional parameter is missing, the default value is 40. For a faster initialization, this parameter should be close to the duration of the input frames. The start/stop of the waveform is reached when the absolute value of two consecutive samples is smaller than the maximum value divided by 1000.

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.

4.4.36. FDIN_S_INJECTION_FLAG

This key trigger the production of an integer channel having the value of one when there are no injections and zero around the time of the injection taken from the FrSimEvent time.

- Parameter 1 (string): name of the produced channel.
- Parameter 2 (integer): sampling rate of the produced channel.
- Parameter 3 (real): time before the injection during which the value is set to zero.
- Parameter 4 (real): time after the injection during which the value is set to zero.

4.4.37. 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.

4.4.38. 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.

4.4.39. 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): optional name of the injection channel. If this parameter is missing, the injection channel is not recorded.
- Parameter 4 (string): name of the FrSimEvent 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.

4.4.40. FDIN_SEGMENTS

This keyword let you apply segments list on a data quality channel. The value of the dataQuality channel is set to 0 (meaning bad) when the GPS time is outside the segments listed in the segments file.

- Parameter 1 (string): Channel name on which the segments are applied. This should be a channel of 32 bit integers.
- Parameter 2 (string): Segments file name. This is an ASCII file containing a list of start and stop GPS time for each segment (two numbers per line). The GPS time need to be larger than 6.e8. GPS time could be floating point numbers.

Remark: as an example, a 100 Hz quality channel named `L1:MBTA_CAT1` with its value always set to 1 could be created using the key:

```
FDIN_ADD_SURROGATE_DQ_CHANNEL L1:MBTA_CAT1 100 0 1 1.e10 0 1 0
```

4.4.41. 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.

4.4.42. FDIN_SELECT_FRAMES

Same key as FDOUT_SELECT_FRAME. This key is mostly useful at output/write time, but could be used at input time for testing frames discontinuity in a pipeline for instance.

4.4.43. FDIN_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.

4.4.44. 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 enables 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.

4.4.45. FDIN_TREND_OF_TREND

This key let you build trend frame at lower rate.

- Parameter1 (dec): New trend period in seconds like 100.

Remarks:

- The time series for all channels with the suffix “_min”, “mean” or “_max”, will be replaced by new time series with the new period, taking the minimum, mean or maximum values over one new period.
- Channels with the suffix “_rms” or with more than one dimension will be removed.
- The other channels with more than one value par period will be converted to “_min”, “_mean” and “_max” channels with one value per period.
- The frame duration is unchanged by this key.

4.4.46. FDIN_VETOS

This key works as the FDIN_SEGMENTS key, except that the value of the dataQuality channel is set to 0 (meaning bad) when the GPS time is INSIDE (and not outside) the segments listed in the segments file.

4.5. Keywords to control the frame output(s)

4.5.1. FDOUT_ADD_PREFIX

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

4.5.2. FDOUT_ADD_SUFFIX

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

4.5.3. FDOUT_BUT_FILTER

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

4.5.4. FDOUT_BITOP

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

4.5.5. FDOUT_DOWN_SAMPLING

Same object as FDIN_ DOWN_SAMPLING, but applied at the output time.

4.5.6. FDOUT_DUPLICATED_CHANNELS_CHECK

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

4.5.7. FDOUT_DUPLICATED_CHANNELS_REMOVE

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

4.5.8. 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 criteria
- Parameter 4 (real): Maximum space (in GB) used by all files or zero if not used as criteria
- Parameter 5 (integer): Maximum time span (in seconds) used by all files or zero if not used as 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 that might have been removed manually. This scan determines the number of frame files, the time span covert by the files (information is 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 limits 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

4.5.9. 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 parameter. 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.

- Parameter 4 (int): This "eventMode" optional parameter defines if the columns for events time are filled in the ffl. If eventMode = 0 (default) then the file start/end times are duplicated for the events. If eventMode = 1, then zeros are put, which will slow down the ffl use.

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.

4.5.10. 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.

4.5.11. 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 channels after it is gone. If this is set to zero, the list of channels is only built 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 used 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 keys 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 channels (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 messages in the log files.

4.5.12. FDOUT_COMBINE_CHANNELS

Same as FDIN_COMBINE_CHANNELS but applied at output stage.

4.5.13. FDOUT_COMBINE_FLAGS

Same as FDIN_COMBINE_FLAGS but applied at output stage.

4.5.14. 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 threads 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

4.5.15. FDOUT_CONSUMER

This is an obsolete keyword replaced by FDOUT_CM

4.5.16. FDOUT_CONVERT_NTCT

Same object as FDIN_CONVERT_NTCT but apply at the output time.

4.5.17. FDOUT_CONVERT_SERDATA

Same object as FDIN_CONVERT_SERDATA but apply at the output time.

4.5.18. FDOUT_DELAY

This key should be use to delay (buffer) the frames before outputting them.

- Parameter 1 (integer): The delay expressed as number of frames. Should be multiply by the frame length to get the effective delay.

4.5.19. FDOUT_FFT_FILTER

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

4.5.20. 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 parameter 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
- If the output directory does not exist, it is automatically created at program start.
- The first parameter should not end with a "/" in order to be able to extract a prefix for the file name.

4.5.21. FDOUT_FILE_CHECKSUM

This key redefines the rule for computing the checksum for the last output file. It must be after a FDOUT_FILE key. It can be reuse multiple times to redefine different checksum computation rule for different output files. By default, i.e. if this key is missing, the file and frame checksums are computed.

- Parameter 1 (integer): checksum computation flag. If this parameter is zero, no checksum is computed when writing the file(s). 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 frames and structures checksum are computed. To compute all checksums, use "3".

4.5.22. 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.

4.5.23. FDOUT_FRAME_DURATION

Same object as FDIN_FRAME_DURATION, but applied at output time.

4.5.24. FDOUT_GET_BRMS

Same object as FDIN_GET_BRMS, but applied at output time.

4.5.25. FDOUT_GET_LINE

Same object as FDIN_GET_LINE, but applied at output time.

4.5.26. FDOUT_GET_LINES

Same object as FDIN_GET_LINES, but applied at output time.

4.5.27. FDOUT_HISTORY

Same object as FDIN_HISTORY, but applied at output time.

4.5.28. FDOUT_RECAST_CHANNELS

Same object as FDIN_RECAST_CHANNELS, but applied at output time.

4.5.29. FDOUT_RECOLOR_CHANNELS

Same object as FDIN_RECOLOR_CHANNELS, but applied at output time.

4.5.30. FDOUT_RENAME_CHANNEL

Same object as FDIN_RENAME_CHANNEL, but applied at output time.

4.5.31. FDOUT_REJECT_EVENTS

Same object as FDIN_REJECT_EVENTS, but applied at output time.

4.5.32. FDOUT_RESCALE_CHANNEL

Same object as FDIN_RESCALE_CHANNEL, but applied at output time.

4.5.33. FDOUT_SEGMENTS

Same object as FDIN_SEGMENTS, but applied at output time.

4.5.34. FDOUT_SELECT_CHANNELS

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

4.5.35. FDOUT_SELECT_FRAMES

This key let you reject frames on the basis of their GPS time. This is useful to write only a fraction of the input frames.

- Parameter 1 (int): period used to make the GPS time selection.
- Parameter 2 (int): start of the selection modulo the period.

- Parameter 3 (int): end of the selection modulo the period.

Examples:

- FDOUT_SELECT_FRAMES 200 0 100 will keep processing (i.e. execute all keys after this one) all frame with a GPS time such that 0 <= gps_200 < 100 where gps_200 is the GPS time modulo 200.

Remark: if a FDOUT_STAT key is used; it is better to put the FDOUT_SELECT_FRAMES key after the FDOUT_STAT key to have the userinfo message updated at every frame.

4.5.36. FDOUT_SET_NEXT_STOP

This optional key, which has only one integer parameter, lets redefine the default period (1 second) the process waits 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.

4.5.37. FDOUT_SET_PROC_TYPE

Same object as FDIN_SET_PROC_TYPE, but applied at output time.

4.5.38. FDOUT_SHM

With this key, frames will be written in a circular buffer. The frames files contain only one frame. It is a convenient replacement of the combination of the FDOUT_FILE and FDOUT_CLEAN_DIR keys.

- 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. If the path finish by "$AUTO_PREFIX$", then the is expended to the CmName/CmName.
- Parameter 2 (string): Tag: a selection of channels to be kept in the output frames (see the FDOUT_CM key for the tag definition).
- Parameter 3 (integer): Time span (seconds) of the circular buffer.
- Parameter 4 (integer): This optional parameter defines the period (in seconds) for the removing of the old files. If this argument is not provided or set to zero, the cleaning is performed each time a new file is written.

Remark: with this key, the frame file checksums are all turned off, since the writing is performed in the computer memory and not on an external device.

4.5.39. FDOUT_SKIP_FRAMES_WITHOUT_EVENT

When this key is reached, the output processing stops if the frame does not contain an FrEvent or FrSimEvent. This is useful to write a file with only the frames with events.

4.5.40. FDOUT_STAT

Same object as FDIN_STAT, but applied at 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

4.5.41. FDOUT_THREAD

This key trigger the creation of a new thread for all following output actions. Frames are passed to this thread through a circular buffer. If this buffer becomes full, the main wait until a free slot is available (frames are lost before version v8r35p1). This key could be used multiple times. But, since Cm connection could only be handled in the "main", any Cm send must be performed from the main list of action before the first thread is declared.

- Parameter 1 (integer): The maximum number of frames that could be stored in the circular buffer.
- Parameter 2 (string): A name used for debugging

4.5.42. FDOUT_TO_ACL

This key allows sending ADC values to an Acl process to update a constant using the message of type AcConstChSet (see the Acl package documentation for more details). There is one message sent at each frame for each FDOUT_TO_ACL key. The ramp time is set to 0.

- Parameter 1 (string): Name of the Acl process
- Parameter 2 (string): Name of the Acl constant to be updated

- Parameter 3 (string): Name of the frame ADC channel to be used. If the channel sampling rate is more than one Hz, the mean value over one frame is send. If the channel sampling period is less than the frame duration, the same value is sent at each frame, until a new value is available.
- Parameter 4 (string): Optional value that is “load” or “apply”, according to the Acl documentation. Usually, the first FDOUT_TO_ACL keys use “load” and the last should use “apply”.

4.5.43. FDOUT_TREND_FRAME

This key enables the production of trend frame. A trend frame captures the min, max, mean and rms values of all FrAdcData and FrProc channels (1 d channel of type timeSeries) 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 the trend frame is build
- Parameter 2 (integer): The trend frame length in seconds
- Parameter 3 (integer): This is an optional parameter. If present and set to “-64”, the trend data are stored as 64 bits floating point numbers. Otherwise, the default is to store them as 32 bit floating point numbers.

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.

4.5.44. FDOUT_TREND_OF_TREND

Same object as FDIN_TREND_OF_TREND, but applied at output time.

4.5.45. FDOUT_VETOS

Same object as FDIN_VETOS, but applied at output time.

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 4.5.32).

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

Remark: this section is a bit obsolete; check the latest code version from https://git.ligo.org/virgo/virgoapp/Fd.

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.

There are a few prerequisites on the system configuration:

- The byacc and flex tools need to be installed (this is required by the CSet package).
- On some OS, you need to create a link /usr/bin/gmake that point to /usr/bin/make.
- Parallel compilation is not compatible with the CMT compilation. You should check that the environment variable MAKEFLAGS do not have the option -j set.

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/v4r0
  ```
- Then build the package using CMT

```
cd v4r0/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/v4r0/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 v8r25
```

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.
Notice that the version v8r25 is given as an example. A more recent version might be available.

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

See old doc (up to v8r20) for early history, then the SVN log file and finally the gitlab history.