Frame Library (Fr)
User's Manual

VIR-MAN-LAP-5400-103

Version 8.41

January, 2021

Summary:

Introduction

A frame is a unit of information containing all the information necessary for the understanding of the interferometer behavior over a finite time interval which integrates several samplings. It contains thus not only the sampling performed during the integrated time interval, but also those performed at a frequency smaller than the frame frequency.

To simplify its manipulation, a frame is organized as a set of C structures described by a header holding pointers to additional structures and values of parameters expected to be stable over the integrated time interval: the starting time of the frame, its duration, values produced by the slow monitoring. This header is followed by an arbitrary number of additional structures, each holding the values of a rapidly varying parameter like the main signal, the seismic noise, etc...

This frame structure is a standard which has to be conserved over the various stages of the analysis. Thus Frame history, detector geometry, trigger results, monitoring data, reconstructed data, simulation results just lead to additional structures. It is always possible to add new structures or to drop old ones.

This standard format is the one used by the LIGO and VIRGO Gravitational Wave Detectors. This Document described the software used to manipulate the frames. The definition of the various structures as well as their representation on tape is described in specification document.

The C structures used by The Frame Library

The data are stored in a set of C structures described in the document *Specification of a Common Data Frame Format for Interferometric Gravitational Wave Detector (IGWD)* (VIR-067-08 and LIGO-T970130-v1). The variable names of the C structures are exactly the one given in this document.

A quick tour of the Library: the examples

Many examples are provided with the frame library in the directory src. They have been designed to test the various parts of the library and are good starting points for a new program. The Files are :

- **exampleCompress.c** This program creates a frame with all types of vectors, write it with all the compression types available and read it back to check if the frame has not been corrupted.
- **exampleCopyFile.c** a simple copy file program. See the utility FrCopy for more complex file copy.
- **exampleCopyFrame.c** a simple copy file program with some channel selection. See the utility FrCopy for more complex file copy.
- **exampleFileDump.c** dump on the screen a short summary of a frame file content See the FrDump utility for more options.
- **exampleFull.c** This example builds frames with several ADC and many different types of data. Then the frames are written in a file. Finally, the tag functionality is tested.
- **exampleMark.c** This function shows the use of random access in a frame file.
- **exampleMultiR.c** Reads the various files produced by exampleMultiW.c This program is useful to search for memory leaks.
- **exampleMultiTOC.c** Reads the various files produced by exampleMultiW.c using random access. This program is used to test random access and to search for memory leaks.
- **exampleMultiW.c** Produces several frames in different files. This program is useful to search for memory leaks.
- **exampleOnline.c** creates a few frames and write them in memory. Different compression type could be used to test the speed.
- **exampleReshape.c** This program shows how to change the frame length using the FrReshape functions..
- **exampleSpeed.c** This program test the in memory read/writing speed for a frame given by the user and the a given compress type.
- **exampleStat.c** Illustrates the use of static data.

If you just want to read data from a frame file:

In the simplest case, to extract data from a frame file, you need to use only three FrameL functions:

```
        iFile = FrFileINew("inputFileName");
```

This function will open the frame file. Usually inputFileName is an "ffl", a list of .gwf file which make the data management much easier.

```
        FrVect* vect = FrFileIGetVectF(iFile,  channel_name, tStart,
tLength);
```

which return a vector for "channel_name" starting at the GPS time tStart and for a duration "tLength" . The start time could be in the middle of a frame and the duration could be over multiple frame or files. The vector is of type single floating point, whatever was the storage on file. The number of elements is vect->nData. The data values could be access by vect->dataF[i].
Then you need to free the space by

```
        FrVectFree(vect);
```

If you want to write data in a frame file, here is a simple program:

```
#include "FrameL.h"

int main() {
 FrFile *oFile;
 FrameH *frame;
 FrProcData *proc;
 double sampleRate;
 long nData,i,j;

  frame = FrameNew("demo"); /*----------------------create a 10s long 
frame--*/
  frame->dt = 10;

  sampleRate = 16384;       /*-----add a 16384Hz 32bits float proc channel-
--*/
  nData = sampleRate*frame->dt;
  proc = FrProcDataNew(frame,"Channel_Name",sampleRate, nData, -32);

  /*-------- open output file; compression type 9, 1000 seconds per file --
--*/

  oFile = FrFileONewM("test", 9, "FrFull", 1000);
  if(oFile == NULL) {
    printf("Open file error (%s)\n",FrErrorGetHistory());
    return(0);}

  for(i=0; i<10; i++) {          /*---------------------produce 10 frames -
--*/
      frame->GTimeS = frame->GTimeS + frame->dt;

      for(j=0; j < proc->data->nData; j++) { /*----- update channel 
content--*/
           proc->data->dataF[j] = j;}        /*-- change this to your need-
--*/

      if(i < 2) FrameDump(frame, stdout,2); /*------------- just for debug-
--*/

      if(FrameWrite(frame, oFile) != FR_OK) {
        printf("Write error; last error:%s\n",FrErrorGetHistory());
        return(0);}
     }

  FrFileOEnd(oFile); /*------------------------------ close the output 
file---*/

  FrameFree(frame);

  return(0);
}
```

The Frame Utilities: FrCopy, FrDump and FrCheck

Three utilities are included in the Frame Package.

To copy a (set of) frame(s): FrCopy

- This program reads frames from one or more input files, merge them if requested and write them to one or more output file, using or not data compression. See the help function bellow for program use.
- The syntax is: FrCopy *options*
  where *option* could be:

-i <input file(s)> If more than one files is given after the keyword -i they will be read in sequence. If several input stream are defined (several -i followed by name(s)), then the frame content will be merged
-o <output file>
-f <first frame: (run # frame #) or (GPS time)> Example: -f 0 20 will start with run 0 frame 20. -f 6544444 will start at GPS time = 6544444. If this option is used, the Table Of Content is mandatory and all frames will be read by increasing time.
-l <last frame: (run # frame #) or (GPS time)>; the -l argument is interpretated as the length if it is a small number (<100000000) instead of the GPS time for the last frame.
-c <compression type> Compression types are -1 (same compression as input file), 0 (no compression), 1 (gzip), 3 (differenciation+gzip), 5 (gzip for float+zero suppress for int), 6 (zero suppress for int). The default is 6.
-a <list of input adc channels>.When this option is used, random access are performed to read only the requested adc channels. Only the Frame header is returned in addition to the adc data. Additional information like the history records is not returned.
-t <list of tag channels> Tag is a channel list with wild cards like: ADC122 ADC5* If a name start by - it is interpreted as an anti-tag
-r <new frame length in second> The reshape option works only with one output file.   It assumes that the length of the input frame is a integer number of second. The starting GPS time of the output frame will be a multiple of the frame length. The requested length should be larger than the input frame length.
-decimate <number of sample to average> The decimation is done on all selected channel by doing a simple data averaging.
-d <debug level> (from 0 to 5)
-max <maximum output file length in second>. If this option is used, the output file is split in several file, each of them lasting up to <max> second. The name of these files is no more just <output file> but '<output file>-GPS-maxLength.gwf' (like V-R-730123000-100.gwf if <output file> = 'V-R').
-noTOCts to not write TOC for time series
-noChkSum to not put checksum in the output file.
-h (to get the help)

To dump frames: FrDump

- This program produces a dump of one file, one or more frame or one or more channels.
- The syntax is: FrDump *options*
  where *option* could be:

-i <input file(s)> If more than one files is given after the keyword -i they will be read in sequence. If several input stream are defined (several -i followed by name(s)), then the frame content will be merged
-f <first frame: (run # frame #) or (GPS time)> Example: -f 0 20 will start with run 0 frame 20. -f 6544444 will start at GPS time = 6544444
-l <last frame: (run # frame #) or (GPS time)>
-t <list of tag channels> Tag is a channel list with wild cards like: ADC122 ADC5* If a name start by - it is interpreted as an anti-tag
-d <debug level> (from 0 to 5)
-top <number of ADC in the hit-parade>
-h (to get this help)
If one of the next option is there, we do only a partial frame dump

-adc   to dump only the FrAdcData information
-sms   to dump only the FrSerData information
-proc to dump only the FrProcData information
-sim   to dump only the FrSimData information
-sum   to dump only the FrSummary information
-stat to dump only the static information
-raw   to dump only the raw data information
-event to dump only the FrEvent and FrSimEvent

To check a frame file: FrCheck

This program check that the frame file could be read successfully. The file checksum is also checked if they are available. In case of success, this program returns zero and set the environment variable FRCHECK_NFRAME to the number of frames in the file. It returns an error flag in case of error. By default (unless the -t or -s option are used), FrCheck do first a sequentiel read to check the file checksum, then do a random access read to check the frame checksum.

- The syntax is: FrCheck *options*
  where *option* could be:

  -i <input file> Only one file should be used
  -d <debug level> (default 1). 0 will supress all info and error messages.
  -t to scan the file using only the TOC
  -s to scan the file using only sequentiel read(TOC not used)
  -f GPS time of the first frame to scan (default=0) (only used when doing the random access)
  -l GPS time of the last frame to scan (default : 999999999.) (only used when doing the random access).
  -c to check the data compression (if any)
  -h to get the help

Reference Part

Library control

The Frame library do not need any initialization. However, you can change some of the default parameters using the following function or you can access to some information.

FrLibIni

- This function changes the debug level and output file.
- Syntax: **FILE *FrLibIni(char *outFile, FILE *fOut, int dbglvl)**
  - outFile is the output file name provided by the user
  - fOut should be used only if the user wants to send out the debug information on an already opened file. Then he should provide this pointer. In this case, outFile should be NULL**.**
  - dbglvl is the debug level provided by the user. This is used only for internal and technical debug. Usually you can use 0.
    - 0 means no output at all
    - 1 gives a minimal description
    - 2,3 give more information

- The return argument is the pointer to the file opened. If the output file could not be opened the debug information is sent on stdout and the return argument is stdout. In case of severe error due to insufficient space, the return value is NULL and the user should not go further.

FrLibSetLvl

- This function changes the debug level. It could be called at any time.
- Syntax: **void FrLibSetLvl (int dbglvl);** where: dbglvl is the debug level provided by the user with the same meaning as in FrLibIni.

FrLibVersion

- This function returns the Library version as a float.
- Syntax: **float FrLibVersion (FILE *fOut);** It returns the version number (like 3.70). If fOut is not NULL it print also on fOut some debugging information.

FrLibVersionF

- This function returns the full Library version and compile time as a char*.
- Syntax: **char *FrLibVersionF ();** It returns the version number like "Fr Version:6.07 (May 20, 03)(Compiled: May 20 2003 17:45:54)".

Frame Handling

FrameCompress,	FrameMerge	FrameReshape
FrameCopy,	FrameRead,	FrameTagXXX,
FrameDump,	FrameReadN,	FrameUntagXXX,
FrameDumpToBuf	FrameReadRecycle	FrameWrite
FrameExpand,	FrameReadT	FrameWriteToBuf
FrameFree,	FrameReadTAdc,	FrameRemoveUntaggedData
FrameFindVect	FrameReadFromBuf	Back to summary

FrameCompress

- This function compress in memory all the frame vectors.
- Syntax: **FrameCompress (FrameH *frame, int compress, int gzipLvl)** were:
  - frame is the pointer to the frame header provided by the user
  - compress is the type of compression:
    - 1 for gzip,
    - 3 for differentiation and gzip.
    - 5 for differentiation and zeros suppress (only for short)
    - 6 for differentiation and zeros suppress for short and int, gzip for other
    - 7 for differentiation and zeros suppress for short, int and float to integer (not part of the current frame format)
    - 8 for differentiation and zeros suppress for short, int and float. (not part of the current frame format)
    - 255 for user defined compression code (definitely not part of the frame format)

  - gzipLvl is the gzip compression level (provided by the user). 0 is the recommended value.

- In normal use, the compression is done at frame write and the user do not need to take care of it.

FrameCopy

- This function copy a full frame and return the pointer to the new FrameH structure. This means it allocate the memory for the full tree of structures. The input frame is unchanged. It returns NULL in case of error (memory allocation failed).
- Syntax: **FrameH* FrameCopy (FrameH *frame)** were frame is the pointer to the frame header provided by the user

FrameDump

- This function produce a readable dump of the frame content.
- Syntax: **void FrameDump (FrameH *frame, FILE *fp, int debugLevel)** were:
  - frame is the pointer to the frame header provided by the user
  - fp is the file pointer where the debug information will be send (like stdout)
  - debugLevel is the debug level provided by the user. 0 means no output at all, 1 gives a minimal description (<5 lines per frame), 2, 3 give more information

FrameDumpToBuf

- This function produce a readable dump of the frame content.
- Syntax: **char* FrameDumpToBuf (FrameH *frame, int level, cha* buf, long bufsize)** were:
  - frame is the pointer to the frame header provided by the user
  - debugLevel is the debug level provided by the user. 0 means no output at all, 1 gives a minimal description (<5 lines per frame), 2, 3 give more information
  - buf is a buffer provided by the user
  - bufSize is the buffer size in bytes (provided by the user).
- This function returns the pointer to the beginning of the printable area of buf.

FrameExpand

- This function uncompressed all the frame vectors:
- Syntax: **void FrameExpand (FrameH* frame)** where frame is the pointer to the frame header provided by the user
- In normal use the uncompression is done by a FrameRead call or by the channel access.

FrameFree

- This function all the space allocated for a frame.
- Syntax: **void FrameFree (FrameH* frame)** where frame is the pointer to the frame header provided by the user

FrameGetAdcSize

- This function returns the memory used by a all the ADC structures and associated vectors (in bytes)
- Syntax: **FRLONG FrameGetAdcSize (FrameH *frame)**

FrameFindXXX

- Syntax:
  FrDetector *FrameFindDetector(FrameH *frame, char *detNameOrPrefix)
- This function returns the pointer to the detector structure given its name or 2 letters prefix. It return NULL if the detector is not found.Since the detector is still attached to the frame, the user should NOT free it.

FrameFindXXX

- Syntax:
  FrVect* FrameFindVect (FrameH* frame, char* name)
  FrVect* FrameFindAdcVect (FrameH* frame, char* name)
  FrVect* FrameFindProcVect (FrameH* frame, char* name)
  FrVect* FrameFindSimVect (FrameH* frame, char* name)
  FrVect* FrameFindStatVect (FrameH* frame, char* name)
  FrVect* FrameFindSumVect (FrameH* frame, char* name)
- These functions returns the pointer to the vector associated to one channel (Adc, Proc or FrSimData). FrameFindVect look for all kinds of channels. Name is the channel name. It return NULL if the channel is not found.Since the vector is still attached to the frame, the user should NOT free the vector.

FrameMerge

- This function merges the data of two frames.
- Syntax: **FrameH* FrameMerge (FrameH* frame1, FrameH* frame2)** where frame1 and frame2 are the pointers to the frame headers provided by the user. Data from frame2 are added to frame1. All remaining unused structures of frame2 are deleted.
- No check is performed on the time compatibility. If the timing information is available (GTimeS != 0) then the two times are compared and the possible time residual is stored in the adc->timeOffset variables.

FrameNew

- This function create a new frame: the frame header and the FrDetectProc structure. The local time is used to fill the Frame header timing section. A detector (Proc) structure is also added in order to be able to add right away the static data structure.
- Syntax: **FrameH* FrameNew (char *name)** where name is the experiment name.
- This function returns the pointer to the frame header or null in case of problem.
- Remark: the FrameHNew function (syntax: FrameH* **FrameHNew (char *name)**) creates only a frame header and do not fill the timing information.

FrameRead

- This function read the next frame in a file. It returns NULL in case of error or end of file.
- Syntax: **FrameH* FrameRead (FrFile *iFile)** where iFile is the pointers to the input file provided by the user.
- If you want to not uncompress the data at read time see FrFileSet.

FrameReadN

- This function read the frame starting for a given run and frame numbers. This is a random file access and requires frame files version 4 at least. It returns NULL in case of error or if the frame is not in the file or if the table of content is not available.
- Syntax: **FrameH* FrameReadT (FrFile *iFile, int runNumber, int frameNumber)** where iFile is the pointers to the input file provided by the user.</ NULL in case of error or if the frame is not in the file or if the table of content is not available.
- Syntax: **FrameH* FrameReadT (FrFile *iFile, double gtime)** where iFile is the pointers to the input file provided by the user and gtime the request GPS time. The selected frame is the one which start at gtime or which include gtime. If gtime = 0 then the first frame in the file is returned.

FrameReadTAdc, FrameReadTProc, FrameReadTSer, FrameReadTSim

- This function read the frame starting at a given time with only a defined list of Adc, or Proc or Ser or Sim channels. This is a random file access and requires frame files version 4 at least. It returns NULL in case of error or if the frame is not in the file or if the table of content is not available.
- Syntax:
  FrameH* FrameReadTAdc (FrFile *iFile, double gtime, char *name)
  FrameH* FrameReadTProc (FrFile *iFile, double gtime, char *name)
  FrameH* FrameReadTSer (FrFile *iFile, double gtime, char *name)
  FrameH* FrameReadTSim (FrFile *iFile, double gtime, char *name)
  where
  - iFile is the pointers to the input file provided by the user and gtime the request GPS time (several files could be used at the same time).
  - The selected frame is the one which start at gtime or which include gtime. If gtime = 0 then the first frame in the file is returned.
  - The selected frame is the one which start at gtime or which include gtime. If gtime = 0 then the first frame in the file is returned.
  - name is a string containing a list of channel names of different type separated by a space which could include wild card (example "*LSC* *EXC")

FrameReadFromBuf

- Syntax: **FrameH *FrameReadFromBuf (char *buf, long nBytes, unsigned short comp);** where comp is the compression level. if comp = -1, no compression/uncompress is performed.

FrameReshapeNew, Add, End

- This set of function is designed to change the frame size, i.e. to group consecutive frame in a single one.
- Three calls are needed:
  - **FrameH *FrameReshapeNew (FrameH *frame, int nFrame, int position);** This function initialize the reshaping. The new frame size will be nFrame longer than the original frame size. The new frame will start at time offset equal to "position" times the length of the initial frame. This function returns a pointer to the new frame (same has the input frame) or NULL in case of problem.
  - **int FrameReshapeAdd (FrameH *new, FrameH *frame);** This function moves the information from the frame "frame" to the frame "new" which should be the output of the function FrameReshapeNew. This function performs a FrameFree of the frame part. It returns 0 in case of success or an error code.
  - **int FrameReshapeEnd (FrameH *new);** This function should be call when all the frame part has been added and before the "new" frame could be used. This function returns 0 in case of success or an error code.

- Remark: The copy utility is a convenient way to change the frame size.
- Example: See the file exampleReshape.c

FrameSetPrefix

- Syntax: **void FrameSetPrefix(FrameH* frame, char* prefix);** h This function set the prefix "prefix" to all channels in this frame.

FrameTagXXX with XXX=Adc, Event, Proc, Ser, Sim, SimEvt, Stat, Sum

- Syntax:

```
void   FrameTag     (FrameH *frame, char *tag);
void   FrameTagEvent(FrameH *frame, char *tag);
void   FrameTagAdc  (FrameH *frame, char *tag);
void   FrameTagProc (FrameH *frame, char *tag);
```

void FrameTagSer (FrameH *frame, char *tag);
void FrameTagSim (FrameH *frame, char *tag);
void FrameTagSimEvt(FrameH *frame, char *tag);
void FrameTagStat (FrameH *frame, char *tag);
void FrameTagSum (FrameH *frame, char *tag);

- These functions select all the Adc, Ser, ... which match the names given in the tag string. This string could contain several names, some of them could include "*" and then are interpreted has wild card. After a call to FrameTagXXX all other channels are hidden for the user. If a FrameDump or FrameWrite is performed, only the tagged channels will be dumped or written. For instance the call to:

  void FrameTagAdc(myframe, "Lr* SaDb2")

  will keep from the frame myframe the SaDb2 ADC and all ADC with a name starting with Lr.
  - The function FrameTag call all the other functions. It performed a 'global tag'
  - Remarks:
    - No change of the channel list should be done when the list as been tagged.
    - Wild card "*" could be used anywhere in a name.
    - The tag "*" means select all the channels.
    - The tag "NONE" means no channels are selected.
    - A tag starting by "-" is interpreted as an 'anti-tag': the channel is removed.
    - It is not necessary to call FrameUntag before doing a FrameFree.

FrameUntagXXX with XXX=Adc, Event, Proc, Ser, Sim, SimEvt, Stat, Sum

- Syntax:
  void FrameUntag (FrameH *frame);
  void FrameUntagAdc (FrameH *frame);
  void FrameUntagEvent(FrameH *frame);
  void FrameUntagProc (FrameH *frame);
  void FrameUntagSer (FrameH *frame);
  void FrameUntagSim (FrameH *frame);
  void FrameUntagSimEvt(FrameH *frame);
  void FrameUntagStat (FrameH *frame);
  void FrameUntagSum (FrameH *frame);
- These functions restore the channel linked lists.
- The function FrameUntag call all the other function. It performed a 'global tag'

FrameRemoveUntaggedXXX with XXX=Adc, Event, Proc, Ser, Sim, SimEvt, Stat, Sum

- Syntax:
  void FrameRemoveUntagged (FrameH *frame);
  void FrameRemoveUntagAdc (FrameH *frame);
  void FrameRemoveUntagEvent(FrameH *frame);
  void FrameRemoveUntagProc (FrameH *frame);
  void FrameRemoveUntagSer (FrameH *frame);
  void FrameRemoveUntagSim (FrameH *frame);
  void FrameRemoveUntagSimEvt(FrameH *frame);
  void FrameRemoveUntagStat (FrameH *frame);
  void FrameRemoveUntagSum (FrameH *frame);
- These functions remove (free) all the channels which are not tagged. After a call to this function, the FrameUntag function will have no effect;

FrameWrite

- Syntax: **int FrameWrite (FrameH *frame, FrFile *oFile);**

- where:
  - frame is the pointer to the frame Header to be written
  - oFile is the pointer to the output file.
- This functions returns 0 in case of success or an error code in case of failure.

FrameWriteToBuf

- Syntax: **long FrameWriteToBuf (FrameH *frame, unsigned short comp, char *buf, long nBytes, int computeChecksum));**
- where:
  - frame is the pointer to the frame Header to be written
  - comp gives the compression algorithm used at writing time (see FrFileONew). If comp < -1, the checksum is computed when writing the frame in memory and the compression level used is the absolute value of comp.
  - buf is the buffer which will receive the frame
  - nBytes is the buffer size
  - computeChecksum could have the folowing values:
    - 0: no checksum are computed when writing the frame
    - 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".
- This function returns the number of bytes written or 0 in case of error.

FrAdcData: ADC's data manipulation

FrAdcDataDecimate

- This function reduces the sampling frequency of and ADC by averaging nGroup bins together (in this version, no filter is performed). It increases the number of bits of the appropriate number of nGroup is positive or keep the original number of bits of nGroup is negative. This version works only for short, int, float and double.
- Syntax: **int FrAdcDataDecimate (FrAdcData *adc, int nGroup)**
- This function returns 0 in case of success and a non zero value in case of error.

FrAdcDataDump

- This function produces a formatted dump of the Adc data. The recommended debug level is 2 (with 0 no output is produced).
- Syntax: **void FrAdcDataDump (FrAdcData *adc, FILE *fp, int debugLvl)** were:
  - adc is the pointer to the FrAdcData structure provided by the user
  - fp is the file pointer where the debug information will be sent (like stdout)
  - debugLevel is the debug level provided by the user. 0 means no output at all, 2 gives about 5 lines of information.
- Remark: The DataValid flag is printed according the Virgo use. DataValid = 0 means no problem. The six lower bits are used to describe the following problems:
    1 means non-valid floating point (only used for floating point)
    2 means some data are missing at known position in the vector (see adc->next vector)
    3 means some data are missing at unknown position in the vector
    4 means front end error: hardware parity error (DOL error)
    5 means front end error: too slow DAQ (FIFO full for instance)
    6 means front end error: invalid format
  For example dataValid = 0x14 means FIFO full and missing data at unknown position.

FrAdcDataFind

- This function finds an FrAdcData structure in a frame. It returns the pointer to the FrAdcData or null if the structure does not exist.
- Syntax: **FrAdcData* FrAdcDataFind (FrameH* frame, char* name)**

FrAdcDataFree

- This function free all the space allocated for the FrAdcDat structure and the linked structure. This function should be used only if the FrAdcData has been created outside a frame like when using random access (FrAdcDataReadT).
- Syntax: **FrAdcData *FrAdcDataFree (FrAdcData *adc);**

FrAdcDataGetSize

- This function returns the memory used by an ADC structure and the associated vector (in bytes)
- Syntax: **FRLONG FrAdcDataGetSize (FrVect *vect)**

FrAdcDataNew and FrAdcDataNewF

- These functions allocate the space for the data of an ADC, and attach it to the FrameH structure (including the creation of the FrRawData structure if does not yet exist). They just differ by the number of parameters : FrAdcDataNewF perform a full fill of the FrAdcData structure.
- Syntax:
    - **FrAdcData *FrAdcDataNewF (FrameH *frame, char *name, char *comment, unsigned int channelGroup, unsigned int channelNumber, int nBits, float bias, float slope, char *units, double sampleRate, int nData);**
    - **FrAdcData* FrAdcDataNew (FrameH* frame, char* name, double sampleRate, int nData, int nBits)**
- The parameters (provided by the user) are:
    - frame (FrameH*) is the pointer to the root frameH structure. frame = NULL is a valid option (the FrAdcData is created independently of a frame)
    - name (cha*) is the ADC name. This name should be unique within a frame for all ADC structures.
    - comment (char *) is any comment the users need to add to the adc description (could be NULL).
    - channelGroup (int) is the channel group (usually a ,mix of crate number, slot number)
    - channelNumber (int)
    - nBits (int) is the number of bits used to store the information. The word length will be either 1, 2, 4 (or 8) bytes. A negative values means that we store floating point number (nBits = 12 is store as a short, nBits =-32 is stored in a 4 bytes float).
    - bias. (float) Any known pedestal
    - slope (float). The calibration constant.
    - units (char) The calibration unit
    - sampleRate (double) is the sampling frequency in Hz.
    - nData (int) is the number of data for this ADC within a frame
- In case of problem, the function returns NULL.
- Remark: this function only allocates the space. The user has to fill the adc data vector. For example, to fill the vector with values coded on 2 bytes:

```
for(i=0; i<adc->data->nData; i++)
    {adc->data->dataS[i] = (the adc value);}
```

- It is possible to allocate multidimension vectors, to support images for instance. In that case, the first dimension must be the time.

FrAdcDataReadT

- Syntax: **FrAdcData *FrAdcDataReadT (FrFile *iFile, char *name, double gtime);**

- This function performs a random read access on the file *iFile for a given GPS time (gtime). I gtime = 0 then the data for the first frame in the file is returned. Only the data for the given Adc are read. Several adc name could be requested in name (name should be separate using space). Names could also include wild card. The function returns a pointer to the FrAdcData structure or NULL in case of error (frame not in file, not Table Of Content, malloc failed). It returns also the associated vector but not the associated table.
- After using FrAdcDataRead, the user should free the memory by calling FrAdcDataFree since the FrAdcData structure has been directly extract from a file whitout frame to take care of memory clean up.

FrAdcDataSetDataValid, ..SetFShift, ...SetTOffset

- Syntax:

  void FrAdcDataSetAux (FrAdcData *adc, FrVect *aux);
  void FrAdcDataSetDataValid (FrAdcData *adc, unsigned short dataValid);
  void FrAdcDataSetFShift (FrAdcData *adc, double fShift, float phase);
  void FrAdcDataSetTOffset (FrAdcData *adc, double tOffset);

- These functions set some fields in the FrAdcData structure.

FrDetector

- **void FrDetectorDump (FrDetector *detector, FILE *fp, int debugLvl);** Dump a detector structure.
- **FrDetector *FrDetectorNew (char *name);** Allocate a detector structure
- **void FrDetectorFree (FrDetector *detector);** Free a detector structure and associated data.

FrEvent

- Constructor: **FrEvent *FrEventNew (FrameH *frameH,**
  char *name, char *comment, char *inputs,
  double GTime, float timeBefore, float timeAfter,
  float amplitude, float probability, char *stat,
  FrVect *data, int nParam, ...);
  When nParam is not zero, the event parameters are added right after nParam as a sequence of name[0], value[0], name[1], value[1],... WARNING: the type of the additional parameters needs to be a 'double' even if they are store as 'float'.
  Example of use:
  ```
      event = FrEventNew(frame, "Inspiral","MBTA algorithm with 2.5PN templates","V0:Pr_B1_ACq"
          710123123.44, 10., 0.1, 1.e-21, 5.3, "signal/rms", NULL, 3, "m1", 1.4, "m2", 1.4, "chi2" 3.2);
  ```
- To copy one event (and the associated data if any, but not the linked list): **FrEvent* FrEventCopy (FrEvent *eventl);**
- Dump: **void FrEventDump (FrEvent *event, FILE *fp, int debugLvl);**
- Destructor: **void FrEventFree (FrEvent*event);**
- Find it in a frame: **FrEvent *FrEventFind(FrameH *frame, char *name, FrEvent *last).** Since there could be more than one event with the same name in one single frame, the "last" parameters is used to make the selection. This function will return the next FrEvent structure following the last structure in the linked list and matching the "name". If last = NULL, the function returns the first event.

- To add an event to a frame: **void FrameAddEvent(FrameH *frame, FrEvent *event).** The event(s) (a single one or a linked list) is added at the end of the event linked list of this frame.
- File random access
- To find all the events within a given time range and with some selection on the event amplitude:
  FrEvent *FrEventReadT (FrFile *iFile, char *name, double tStart, double length, double amplitudeMin, double amplitudeMax);
  This function perform a random read access on the file *iFile. It returns all FrEvent structure (as a linked list) which have a time between tStart and tStart+length and with an amplitude in the [amplitudeMin, amplitudeMax] range. It does NOT return the associated vector nor the associated tables (this could be added later on using FrEVentReadData). The string name could contain several names and wild cards. The function returns a pointer to the first FrEvent structure of the linked list or NULL in case of error (frame not in file, not Table Of Content, malloc failed).
- To find all the events within a given time range and with some selection on several parameters.:
  FrEvent *FrEventReadTF (FrFile *iFile, char *name, double tStart, double length, int readData, int nParam, ...);
  the additional parameters are: **char* paramName1, double min1, double max1, char* paramName2, ...)** where the paramName* are "amplitude", "timeBefore", "timeAfter" or one of the extra event parameter

  This function performs a random read access on the file *iFile. It returns all FrEvent structure (as a linked list) which have a time between tStart and tStart+length and with the extra parameters in the required range. The associated vector is read if the readData flag is set to 1 (or not read if set to 0). The string name could contain several names and wild cards. The function returns a pointer to the first FrEvent structure of the linked list or NULL in case of error (frame not in file, not Table Of Content, malloc failed).
  Example of use: event = FrEventReadTF(iFile,"Inspiral*",t0,50.,1, 2, "M1", 2., 3., "M2", 1., 3.); will return the linked list of all events with a name starting by Inspiral, with a time in the t0, t0+5à range, with a parameter M1 (and M2) in the range 2.;3. (1.;3.).

- To read the associated vector for one event:
  int FrEventReadData (FrFile *iFile, FrEvent *event);
- **Parameters handling:**
  - Add one more parameter to an event:
    FrEvent *FrEventAddParam (FrEvent *event, char* paramName, double value);
    This function returns NULL in case of error (bad input parameters of malloc failed).
  - Add one vector parameter to an event:
    int FrEventAddVect (FrEvent *event, FrVect* vect, char* newName);
    int FrEventAddVectF (FrEvent *event, FrVect* vect, char* newName);
    This function attach a copy of a vector (cast to a vector of float for ...AddVectF) to an event. If newName is not NULL, the vector name is changed. It returns 0 in case of success.
  - Get the value for one parameter:
    double FrEventGetParam (FrEvent *event, char* paramName);
    This function returns -1. if the parameter could not be found.
  - Get event the parameter id:
    int FrEventGetParamId (FrEvent *event, char* paramName);
    This function returns the parameter number in the list or -1 if the parameter could not be found. The parameter value could then be access at event->parameters[id].
  - To find the pointer to a vector attached to the event:
    FrVect* FrEventFindVect (FrEvent *event, char* vectName);
    This function returns a pointer to the vector or NULL if not found. The returned vector is uncompressed. The user should NOT free the vector.
  - To return a copy of a vector attached to the event:
    FrVect* FrEventGetVect D(FrEvent *event, char* vectName);
    FrVect* FrEventGetVect F(FrEvent *event, char* vectName);
    This function returns a pointer to a copy of type double (..VectD) or float (...VectF) of a vector or NULL if not found. The user MUST free the vector after its use.
- **To save on standalone event in a file:**
  - **int FrEventSaveOnFile(FrEvent *event, FrFile *oFile);** This function creates the needed frame header. If the event is part of a linked list, only this event is saved on file. It returns 0 in case of succes.

Input File: FrFileI

FrFileIDump

- This function dumps a summary (file name starting time and file length) of a file. This could be used to create Frame File List.
- Syntax : **void FrFileIDump (FrFile *iFile, FILE *fp, int debugLvl, char *tag);**
- If debugLvl = 0, then only the available values of start time are printed. To get the real values, you need to set debugLvl to 2. The 'tag' parameter is used to defined a list of channels for which we require some information (for instance, use tag = "*B1*" to get only the list of channels with a name containing 'B1'.
- Example: FrFileIDump (file, stdout, 1, NULL); will dump on the standard output all the file names and time information.

FrFileIEnd: close a input file

- This function close an input file and free all the associated structures.
- Syntax: **void FrFileIEnd (FrFile *iFile);**

FrFileIGetVect, ...GetV, ...VectF, ...VectFN, ...VectD, ...VectDN:

- These functions provide random access for the vector of a single channel (FrAdcData, FrSimData or FrProcData) called 'name'. The starting GPS time is tStart, the vector length in seconds is length.
- Syntax:
  - **FrVect *FrFileIGetVect(FrFile *iFile, char *name, double tStart, double length);**
  - **FrVect *FrFileIGetVectD(FrFile *iFile, char *name, double tStart, double length);**
  - **FrVect *FrFileIGetVectDN(FrFile *iFile, char *name, double tStart, double length);**
  - **FrVect *FrFileIGetVectF(FrFile *iFile, char *name, double tStart, double length);**
  - **FrVect *FrFileIGetVectFN(FrFile *iFile, char *name, double tStart, double length);**
- The returned vector starts at the requested time and last exactly the requested number of second (this is new since version 5).
- The vector is converted to a vector of floats (type=FR_VECT_4R) for ...GetVectF and ...GetVectFN or double (type=FR8VECT_8R) for ...GetVectD or ...GetVectDN
- The fonctions FrFileIGetVectDN and ...VectFN return a normalized vector using the FrAdcData information.
- FrFileIGetV is the old name for FrFileIGetVect.
- If there are missing frames in the request time stretch, the corresponding bins are filled with the vector mean value and an additional vector is returned attached to the field "next" of the main vector. This additional vector has the same size of the main vector but each bin contains the number of frames found for this sample. A zero is therefore used for missing sample. If there are no missing samples the filed "next" is set to "NULL".
  - **FrVect *FrFileIGetVAdc (FrFile *iFile, char *name, double tStart, double length, int group);**
  - **FrVect *FrFileIGetVSim (FrFile *iFile, char *name, double tStart, double length, int group);**
  - **FrVect *FrFileIGetVProc (FrFile *iFile, char *name, double tStart, double length, int group);**

FrFileIGetXXXNames:

- Syntax:

  FrVect *FrFileIGetAdcNames(FrFile *iFile);
  FrVect *FrFileIGetDetectorNames (FrFile *iFile);

FrVect *FrFileIGetEventNames (FrFile *iFile);
FrVect *FrFileIGetProcNames(FrFile *iFile);

FrVect *FrFileIGetSimNames(FrFile *iFile);
FrVect *FrFileIGetSimEventNames (FrFile *iFile);

FrVect *FrFileIGetStatNames (FrFile *iFile);

- These functions extract channel or event names from the Table Of Content. It returns one vector containing the list of names (vector of char *: FR_VECT_STRING) or NULL in case or error.

FrFileIGetFrameInfo:

- Syntax: **FrVect *FrFileIGetFrameInfo (FrFile *iFile, double tStart, double length);**
- This function extracts frame information from the Table Of Content. The tStart and length arguments could be used to specify a time range. It returns a linked list of three vectors (or NULL in case or error):
  - The Frame GPS starting time (vector of double)
  - The frame length (vector of double)
  - The frame data quality (vector of int)
- There is one entry per frame in these vectors. The frames are sorted by increasing GPS time.

FrFileIGetChannelList:

- Syntax: **char* FrFileIGetChannelList(FrFile *iFile, int gtime);**
- This function allocates and return a string containing the list of channels contained in a file at a given GPS time and additional meta data.
- The user should take care of the memory free.
- If gtime == 0, the channel list is return for the current file position or for the beginning of the file if no frame has been read

FrFileIGetEventInfo and SimEventInfo:

- Syntax:
  FrVect *FrFileIGetEventInfo (FrFile *iFile, char *tag, double tStart, double length,
  double amplitudeMin, double amplitudeMax);
  FrVect *FrFileIGetSimEventInfo (FrFile *iFile, char *tag, double tStart, double length,
  double amplitudeMin, double amplitudeMax);
- These functions extract (simulated) event information from the Table Of Content. The tStart and length arguments could be used to specify a time range as well the minimum and maximum amplitude. The paramter "tag" let you select the events you want. It could contain wilde cards. If tag = "*" then all events are selected. It returns a linked list of two vectors (or NULL in case or error):
  - The event GPS time (vector of double)
  - The event amplitude (vector of float)
- The events are sorted by increasing GPS time.

FrFileINew:

- This function open one or several files.
- Syntax: **FrFile *FrFileINew (char *fileName);** where **fileName** could be:
  - a single file name like "file1.dat"
  - a list of files separeted by space like "file1.dat file2.dat". In that case file1.dat will be first open and when all the frames from this file will by read, it will automatically open the file file2.dat without any special action from the user. It is an easy way to concatenate files. Or to use several files with random access.
  - a **Frame File List.** This is an ASCII file with the file extension ".ffl". which contain either:

        - a plain list of file like
          file1.dat
          file2.dat
          file3.dat
        - a list a file with GPS information for the file start, file length, time of the first event, time of the last event. This is the output of the FrDump tool with the "-d 0" option. So the best way to build the ffl is to issue a command like "FrDUmp -i file*gwf -d 0 > file.ffl". This will give for instance:

```
file1.dat      666000000.000000 11  666000000.200000 666000010.100000
file2.dat      666000011.000000 11  666000011.200000 666000021.100000
file3.dat      666000022.000000 11  666000022.200000 666000032.100000
```

          These full ffl provide efficient random access on a large number of files.
    - Remark: all file name should start by an non digit character.
    - To read frames from a buffer, see the function FrameReadFromBuf
    - If you want to turn off the decompression during frame read you should type after the file opening:

      iFile->compress = 1;

      Then all the vectors will remain compressed.

FrFileINewFd

- This function open an input file for a given file descriptor
- Syntax : **FrFile *FrFileINewFd (FrIO *frfd);**
- This function returns a pointer to the input file or NULL if an error occurs.

FrFileIRewind

- This function rewind to file. The next FrameRead will then return the first frame in the file.
- Syntax : **FrFile *FrFileRewind (FrFile *file);**
- This function returns a pointer to the input file or NULL if an error occurs.

FrFileISetTime

- This function set the file to a given GPS time. The next frame read will be for this GPS time. If called for a time corresponding to a gap in the file, the reading pointer is set to the next existing frame.
- Syntax : **int FrFileSetTime(FrFile *file, double gpsTime);**

FrFileITFirstEvt, FrFileITLastEvt

- This function returns the GPS time of the first/last event in the file(s). If more than one file is given, it returns the minimum event time and the maximum event time for all the files. These functions work only for files with table of content. They return a negative time in case of error.
- Syntax :

  double FrFileITFirstEvt (FrFile *iFile);
  double FrFileITLastEvt (FrFile *iFile);

FrFileITStart, FrFileITEnd

- This function returns the GPS time of the first/last frame in the file(s). If more than one file is given, it returns the minimum starting time and the maximum end time of all files. They work only for files with table of content. They return a negative time in case of error.
- Syntax :

double FrFileITStart (FrFile *iFile);
double FrFileITEnd (FrFile *iFile);

FrFileITNextFrame

- Syntax : **double FrFileITNextFrame (FrFile *iFile, double gtime);**
- This function returns the GPS time of the next frame (ie the frame starting after gtime). It works only for files with table of content. It returns a negative time in case of error.

Output File: FrFileO

FrFileOEnd:

- To close an output file, you need to call:
  int FrFileOEnd (FrFile *file);

FrFileONewXXX

- This function open an output file for a given name. or file descriptor.
- Syntax:

  FrFile *FrFileONew (char *fileName, int compress);
  FrFile *FrFileONewH (char *fileName, int compress, char *program);
  FrFile *FrFileONewM (char *fileName, int compress, char *program, int maxLength);
  FrFile *FrFileONewMD (char *fileName, int compress, char *program, int maxLength, char* filePrefix, int dirPeriod);
  FrFile *FrFileONewFd (FrIO *frfd, int compress);

- compress gives the compression algorithm used at writing time.
  - -1 to write data without changing the initial compression state
  - 0 for no compression,
  - 1 for gzip (The level of gzip compression could be set by a call to FrFileOSetGzipLevel (file, level) with 0<level<10. The default value is level=1.)
  - 3 for differentiation and gzip.
  - 5 for differentiation and zeros suppress (only for short)
  - 6 for differentiation and zeros suppress for short and gzip for other.
  - 8 for differentiation and zeros suppress for short int and float and gzip for other. (recommended)
- **FrFileONewH** has an extra argument (program) which is string which will be added to the history record at writing time.
- **FrFileONewM** has one more extra parameter: maxLength that define the maximum length for a file in second. When this mawimum is reached, the file is closed and a new one is open. This is convienent to handle large data set. The name of these files is no more just "fileName" but "fileName-GPS-maxLength.gwf " (like V-R-730123000-100.gwf if fileName = "V-R").
- **FrFileONewMD** has two more parameters: filePrefix which defines the file beginning of the file name and dirPeriod which defines the time spam cover by a directory. With these parameters a new file is open each time the GPS time reach a multiple of maxLength and a new folder is created each time the GPS time is a multiple of dirPeriod. If path="./Test", maxLength=100, prefix = "Test" and dirPeriod=1000 then file name will be like: ./Test-800000/Test-800000100-100.gwf).
- When an output file has been open, you can suppress the writing of the Table Of Content for the time series (FrAdcData, FrSimData, FrProcDat, FrSerData, Summary) by setting:

  ```
  oFile->noTOCts = FR_YES;
  ```

FrFileOPutV

- This function writes on or more vectors in a file. It automatically creates a frame and attach an FrProcData channel that own the vector as data member.
- Syntax:

  int FrFileOPutV (FrFile *oFile, FrVect *vect);

- This function returns 0 in case of succes or an error code.
- The GPS time of the vector (vect->GTime) needs to be properly set if more than one vector is put in the file.

FrFileOSetMsg

- This function sets the name used at writing time for the history record.
- Syntax: **void FrFileOSetMsg (FrFile *oFile, char *msg);**
- Remark: if msg = NULL no history message will be added to the file. However, it is advised to always add a history record.

File/structure Checksums

- By default, file and structure checksums are computed when writing a file on disk. Only the structure and checksum are checked when reading a file. Checksums are not computed/check when the frame is written/read in memory.
- The default setting could be changed after the file has been open (FrFileIOpen or FrFileOOpen):
  1. To turn on (or off) the file checksum: iFile->chkSumFiFlag == FR_YES (or FR_NO);
  2. To turn on (or off) the structure checksum: iFile->chkSumFrFlag == FR_YES (or FR_NO);
- The utility FrCheck could be used to verify the file checksums.
- Checksums are available only since version 4.40

FrFilter

- A filter structure as be set up to hold lilnear filter information to be stroed in file. The following function could be used to manage them. See the FrFilter.h file for more details:
  - **void FrFilterFree(FrFilter* filter);**
  - **FrFilter* FrFilterNew(char* name, double fs, double gain, int ntaps, ...);**
    constructor: the additional parameters are ntaps "a" values followed by ntaps "b" values
  - **void FrFilterDump(FrFilter *f, FILE *fp, int debugLvl);**
    This function dumps on file fp (like 'stdout') the filters parameters
  - **FrVect* FrFilterPackToVect(FrFilter *filter);**
    This function creates a vector containing the content of the filter
  - **FrFilter* FrFilterGetFromVect(FrVect *vect);**
    This function creates a filter which was previously packed in a vector
  - **void FrProcDataAddFilter(FrProcData *proc, FrFilter *Filter);**
    this function pack a filter into a vector and attach it to the proc data. The filter structure is untouched
  - **FrFilter* FrProcDataGetFilter(FrProcData *proc, char *name);**
    This function creates a FrFilter structure according to the parameters of the filter called "name" and attached to the proc data

    - **void FrStatDataAddFilter(FrStatData *stat, FrFilter *filter);**
    - **FrStatData* FrameAddStatFilter(FrameH* frame, char* detectorName,char* statDataName, unsigned int tStart, unsigned int tEnd, unsigned int version, FrFilter *filter)**
    - **FrFilter* FrameGetStatFilter(FrameH *frame, char *detectorName, char *statDataName, char *filterName, int gpsTime);**

FrHistory

The best way to add an history record is to used the FrFileONewH function which will set the default history record produced at each FrameWrite to the one you want. However, the FrHIstory records could be manipulated using the following functions.

FrHistoryAdd

- This function adds an history record. A time stamp is automatically added. The string comment is provided by the user. Its format is free. If frame = NULL the history structure is created but not attached to the frame header. These history records are useful to keep track of the various frame processing. This function returns the pointer to the first History structure or NULL in case of malloc error.
- Syntax: **FrHistory *FrHistoryAdd (FrameH *frame, char *comment);**

FrHistoryFree

- This function free the history records and all attached history.
- Syntax: **void FrHistoryFree (FrHistory *history);**

FrMsg

- An online log message could be added to the frame by using the function:
  FrMsg *FrMsgAdd (FrameH *frame, char *alarm, char *message, unsigned int severity);
  The string message as well as the alarm name are provided by the user. Its format is free. The severity value is provided by the user. frame = NULL is a valid option. This function returns the pointer to the FrMsg structure or NULL in case of malloc error.
- To dump it : **void FrMsgDump (FrMsg *msg, FILE *fp, int debugLvl);**
- Find it in a frame: **FrMsg *FrMsgFind(FrameH *frame, char *alarm, FrMsg *last).** Since there could be more than one FrMsg with the same name in one single frame, the "last' parameters is used to make the selection. This function will return the next FrMsg structure following the last structure in the linked list and matching the "name". If last = NULL, the function returns the first found structure.

FrProcData

- These functions have the same meaning as for the FrAdcData structure:
    - Constructor: **FrProcData *FrProcDataNew (FrameH *frame, char *name, double sampleRate, int nData, int nBits);**
    - Dump: **void FrProcDataDump (FrProcData *procData, FILE *fp, int debugLvl);**
    - Destructor: **void FrProcDataFree (FrProcData *procData);**
    - Find it in a frame: FrProcData ***FrProcDataFind (FrameH *frame, char *name)**
    - File random access (FrProcData and vector): **FrProcData *FrProcDataReadT (FrFile *iFile, char *name, double gtime)**

- To add an history record: **FrHistory *FrProcDataAddHistory(FrProcData *proc, char *comment, int nPrevious, ...)**. This function will add an history record containing the comment "comment" and will copy "nPrevious" previous history record(s) from other FrProcData. The additional parameters are the "nPrevious" FrHistory structures. The usage is the following:

  FrProcDataAddHistory(proc, "FFT(V1:Pr_B1_ACq)", 0)   will add only one history record
  FrProcDataAddHistory(proc, "A+B", 2, procA->history, procB->history) will add one history record and will copy the history records from procA and procB where procA and procB are FrProcData structures

- Parameters handling:
- Add a parameter to an FrProcData structure: **FrProcData *FrProcDataAddParam (FrProcData *proc, char* paramName, double value);** This function returns NULL in case of error (bad input parameters or malloc failed).
- Get parameter value: **double FrProcDataGetParam (FrProcData *proc, char* paramName);**
  This function returns -1. if the parameter could not be found.
- Get parameter id: **int FrProcDataGetParamId (FrProcData *proc, char* paramName);**
  This function returns the parameter number in the list or -1 if the parameter could not be found. The parameter value could then be access at proc->auxParam[id].
- To attach a vector to a procData: **void FrProcDataAttachVect(FrProcData *proc, FrVect *vect);**
  Afte this call the vector still belong to the procData and the user must NOT try to free it.
- To find the vector named "name" attached to one procData: **FrVect* FrProcDataFindVect(FrProcData *proc, char *name);**
  After this call, the vector is still own by the proc data (the user must NOT free it).

FrSerData

- Unless specified, these functions have the same meaning as for the AdcData structure:
  - Constructor: **FrSerData *FrSerDataNew (FrameH *frame, char *smsName, unsigned int serTime, char *data, double sampleRate);** The SerData is defined by a name and a GPS time. Usually the data are all included in the data string. The suggest form is to use a string which is a sequence of names and values (for instance "P1 1.e-6 P2 1.e-7").

  Units and serData:

  If in the FrSerData string, there is the keyword "units" followed by a string like "mbar" then this unit will be stored by the DAQ (instead of the default "Count"), put in the units filed of the FrAdcData channel produced by the trend frame builder and then display by dataDisplay when looking at trend data.
  Notice that once you use the keywords "units", this unit will be assigned to all following data. So, if you have a mix of channels, it is better to group them and put the channels without unit at the beginning of the string.
  As an example, the string
  "K 345 units C T1 21.34 T2 25.3 units mbar P1 1013.1"
  Translate to
  K = 345 Counts
  T1 = 21.34 C
  T2 = 25.3 C
  P1 = 1013.1 mbar

  - Dump: **void FrSerDataDump (FrSerData *serData, FILE *fp, int debugLvl);**
  - Destructor: **void FrSerDataFree (FrSerData *serData);**
  - Find it in a frame: FrSerData ***FrSerDataFind (FrameH *frame, char *name, FrSerData *last).** Since there could be more than one FrSerData with the same name in one single frame, the "last' parameters is used to make the selection. This function will return the next FrSerData

structure following the last structure in the linked list and matching the "name". If last = NULL, the function returns the first found structure.

- Find the value of one parameter (smsParama): **int FrSerDataGet (FrameH *frameH, char *smsName, char *smsParam, double *value);** It assumes that the data are stored in the data string as names followed by values for the serial data smsName. It returns 0 in case of success.
- Return directly the parameter value or a default value if not found: **double FrSerDataGetValue (FrameH *frameH, char *smsName, char *smsParam, defaultValue);**
- File random access: **FrSerData *FrSerDataReadT (FrFile *iFile, char *name, double gtime)**

FrSimData

FrSimData *simData, FILE *fp, int debugLvl);

- Destructor: **void FrSimDataFree (FrSimData *simData);**

- Find it in a frame: FrSimData ***FrSimDataFind (FrameH *frame, char *name)**

- File random access (FrSimData and vector): **FrSimData *FrSimDataReadT (FrFile *iFile, char *name, double gtime)**

- File random access (associated vector for one or more frame): **FrVect *FrFileGetVSim (FrFile *iFile, char *name, double tStart, double length)**

FrSimEvent

- Unless specified, these functions have the same meaning as for the AdcData structure:
  - Constructor: **FrSimEvent *FrSimEventNew (FrameH *frameH,**
    char *name, char *comment, char *inputs,
    double GTime, float timeBefore, float timeAfter,
    float amplitude, FrVect *data, int nParam, ...);
    When nParam is not zero, the event parameters are added right after nParam as a sequence of name[0], value[0], name[1], value[1],...
    Example of use:
    Add one vector parameter to an event:**int FrSimEventAddVect (FrSimEvent *event, FrVect* vect, char* newName);**
    int FrSimEventAddVectF (FrSimEvent *event, FrVect* vect, char* newName);
  - This function attach a copy of a vector (cast to a vector of float for ...AddVectF) to an event. If newName is not NULL, the vector name is changed. It returns 0 in case of success.Get the value for one parameter: **double FrSimEventGetParam (FrSimEvent *event, char* paramName);** This function returns -1. if the parameter could not be found.
  - Get event the parameter id: **int FrSimEventGetParamId (FrSimEvent *event, char* paramName);** This function returns the parameter number in the list or -1 if the parameter could not be found. The parameter value could then be access at event->parameters[id].
  - To find the pointer to a vector attached to the event:
    FrVect* FrSimEventFindVect (FrEvent *event, char* vectName);
    This function returns a pointer to the vector or NULL if not found. The user should NOT free the vector.

- To return a copy of a vector attached to the event:
  FrVect* FrSimEventGetVect D(FrEvent *event, char* vectName);
  FrVect* FrSimEventGetVect F(FrEvent *event, char* vectName);
- This function returns a pointer to a copy of type double (..VectD) or float (...VectF) of a vector or NULL if not found. The user MUST free the vector after its use.
- Dump: **void FrSimEventDump (FrSimEvent *simEvent, FILE *fp, int debugLvl);**
- Destructor: **void FrSimEventFree (FrSimEvent *simEvent);**
- Find it in a frame: FrSimEvent***FrSimEventFind (FrameH *frame, char *name, FrSimEvent *last).** Since there could be more than one FrSimEvent with the same name in one single frame, the "last' parameters is used to make the selection. This function will return the next FrSimEvent structure following the last structure in the linked list and matching the "name". If last = NULL, the function returns the first found structure.
- File random access (FrSimEvent and vector): **FrSimEvent *FrSimEventReadT (FrFile *iFile, char *name, double tStart, double length, double amplitudeMin, double amplitudeMax);**
  This function perform a random read access on the file *iFile. It returns all (as a link list) FrSimEvent structure which have a time between tStart and tStart+length and with an amplitude in the [amplitudeMin, amplitudeMax] range. It returns also the associated vector (if any) but not the associated tables. The string name could contain several names and wild card. The function returns a pointer to the first FrSimEvent structure of the linked list or NULL in case of error (frame not in file, not Table Of Content, malloc failed).
- File random access
- To find all the events within a given time range and with some selection on the event amplitude:
  FrSimEvent *FrSimEventReadT (FrFile *iFile, char *name, double tStart, double length, double amplitudeMin, double amplitudeMax);
  This function perform a random read access on the file *iFile. It returns all FrEvent structure (as a linked list) which have a time between tStart and tStart+length and with an amplitude in the [amplitudeMin, amplitudeMax] range. It does NOT returns the associated vector nor the associated tables. The string name could contain several names and wild cards. The function returns a pointer to the first FrEvent structure of the linked list or NULL in case of error (frame not in file, not Table Of Content, malloc failed).
- To find all the events within a given time range and with some selection on several parameters.:
  FrSimEvent *FrSimEventReadTF (FrFile *iFile, char *name, double tStart, double length, int readData, int nParam, ...);
  the additional parameters are: **char* paramName1, double min1, double max1, char* paramName2, ...)** where the paramName* are "amplitude", "timeBefore", "timeAfter" or one of the extra event parameter
  This function perform a random read access on the file *iFile. It returns all FrEvent structure (as a linked list) which have a time between tStart and tStart+length and with the extra parameters in the required range.The associated vector is read if the readData flag is set to 1 (or not read if set to 0). The string name could contain several names and wild cards. The function returns a pointer to the first FrEvent structure of the linked list or NULL in case of error (frame not in file, not Table Of Content, malloc failed).
  Example of use: event = FrEventReadTF(iFile,"Inspiral*",t0,50.,1, 2, "M1", 2., 3., "M2", 1., 3.); will return the linked list of all events with a name starting by Inspiral, with a time in the t0, t0+5à range, with a parameter M1 (and M2) in the range 2.;3. (1.;3.).

FrStatData

A static data is a structure which may stay valid for more than one frame. FrStatData is versioned and is tied to particular detectors and can change from epoch to epochIt is written on file only once. These data stay as long as they are valid compare to the frame time boundary, or as long there is not a new bloc of data with the same name but with a highest version number. In the case of long frames there could be several static data with the same name if they have different starting times which cover the frame duration. The FrStatData do not need to be confined to the time range of the frame in which it is written.

FrStatDataAdd

- This function adds a static data bloc.
  void FrStatDataAdd (FrDetector *detector, FrStatData *sData);

- See also:
  - **int FrameAddStatData(FrameH* frame, char* detectorName, FrStatData *stat);**
    This function attached a static data to a detector structure belonging to this frame.
    If no detector exists for this name, a new one is created.
    If name is NULL, it is attached to the first detector or to a new detector called "Default" is there is no detector structure.
    Any new detector structure is attached to the frame->detectProc list.
  - **FrStatData* FrameAddStatVector(FrameH* frame, char* detectorName, char* statDataName, unsigned int tStart, unsigned int tEnd, unsigned int version, FrVect* vect);**
    This function attached a vector to a static data to a detector structure belonging to this frame.
    If no detector exists for this name, a new one is created.
    If name is NULL, it is attached to the first detector or to a new detector called "Default" is there is no detector structure.
    Any new detector structure is attached to the frame->detectProc list.
  - **int FrDetectorAddStatData(FrDetector* detector, FrStatData *stat);**
    This function attached a static data to a detector structure. The user must NOT free the static data since it will then belong to the detector.
  - **void FrStatDataAddVect(FrStatData *stat, FrVect *vect);**
    This function attached a vector to a static data structure. The user must NOT free the vector since it will then belong to the static data.

FrStatDataDump

- To dump the static data content on the FILE 'fp' (useful values of debugLevel are 1, 2, or 3):
  void FrStatDataDump (FrStatData *sData, FILE *fp, int debugLevel);

FrStatDataFind

- This function finds a static data bloc.
  FrStatData *FrStatDataFind (FrDetector *detector, char *name, unsigned int timeNow);
  timeNow is the time for which we want the static data. If timeNow = 0 then the first static data with that name is return.
- See also:
  - **FrVect* FrameGetStatVect(FrameH *frame, char *detectorName, char *statDataName, char *vectorName, int gpsTime);**
    This function return a copy vector named "vectorName" attached to the static data named "statDataName".
    The user must take car of the vector free to avoid memory leak.
  - **FrStatData *FrameFindStatData(FrameH *frame, char *detectorName, char *statDataName, int gpsTime);**
    This function return the pointer to the static data named "statDataName" and attached to a frame.
    The user must NOT free the structuer after using it.
  - **FrStatData* FrDetectorFindStatData(FrDetector *det, char *statDataName, int gpsTime);**
    This function return the pointer to the static data named "statDataName" and attached to a detector.
    The user must NOT free the structuer after using it.

FrStatDataFree

- This function free the static data bloc including the vectors and all attached static data.
  void FrStatDataFree (FrStatData *sData);

FrStatDataFreeOne

- This function free the static data bloc including the vectors. It returns the pointer to the next bloc in the linked list.
  FrStatData *FrStatDataFree (FrStatData *sData);

FrStatDataNew

- This function creates a new static data bloc.
  FrStatData *FrStatDataNew (char *name, char *comment, char *represent, unsigned int tStart, unsigned int tEnd, unsigned int version, FrVect *data, FrTable *table);
  where:
  - name is the name of this bloc of static data.
  - comment is some user information
  - tStart is the starting time (GPS) of validity for this bloc
  - tEnd is the end time (GPS) of validity for this bloc (tEnd = 0 means no end)
  - version is the static data version number provided by the user
  - data is the data bloc (like a vector) provided by a user.

  * To attach a static bloc to a frame you should attach it to one detector structure.

FrStatDataReadT

- To extract on static data block from a file using a random access read.
  FrStatData *FrStatDataReadT (FrFile *iFile, char *staticDataName, double gpsTime);

Example: Suppose you have

stat_data_1 which is valid in [t1,t1+T1)
stat_data_2 which is valid in [t2,t2+T2)

where t2>= t1 + T1.

Then if your frame file is for time [t0,t0+T0) and you ask for the stat_data at time t in this frame file with t0<= t< t0 + T0 you will get:

stat_data_1 , starting at time t1 , if t1<= t< t1 + T1
stat_data_2 , starting at time t2 , if t2<= t< t2 + T2
nothing otherwise.

Note: The vector should not be time series, or if it is the case, the full vector is returned, ignoring it is a time series, (i.e. independently of the requested start time and effective start time). Therefore FrStatData should not be used to store time series to avoid confusion.

FrStatDataTouch

- When you update the content of a static data bloc you should tell the system by calling:
  void FrStatDataTouch (FrStatData *sData);

FrSummary

- Unless specified, these functions have the same meaning as for the AdcData structure:
    - Constructor: **FrSummary *FrSummaryNew (FrameH *frame, char *name, char *comment, char *test, FrVect *moments, FrTable *table);**
    - Dump: **void FrSummaryDump (FrSummary *summary, FILE *fp, int debugLvl);**
    - Destructor: **void FrSummaryFree (FrSummary *summary);**
    - Find it in a frame: FrSummary ***FrSummaryFind (FrameH *frame, char *name).**
    - File random access (FrSimEvent and vector): **FrSummary *FrSummaryReadT (FrFile *iFile, char *name, double tStart, double length);** This function perform a random read access on the file *iFile. It returns all (as a link list) FrSummary structure which have a time between tStart and tStart+length. It returns also the associated vector (if any) but not the associated tables. The string name could contain several names and wild card. The function returns a pointer to the first FrSummary structure of the linked list or NULL in case of error (frame not in file, not Table Of Content, malloc failed).

FrTable

- Complex tables could be created to stored different types of object. However, tables are not efficient for small numbers of values where a simple string encoding or a plain vector is more efficient.
    - Constructor: **FrTable *FrTableNew (char *name, char *comment, int nRow, int nColumn, ...);**
    - Constructor: **FrTable *FrTableCopy (FrTable *table);**
    - Constructor: **void FrTableExpand (FrTable *table);**
    - Dump: **void FrTableDump (FrTable *table, FILE *fp, int debugLvl);**
    - Access on column: **FrVect* FrTableGetCol (FrTable *table, char *colName);**
    - Destructor: **void FrTableFree (FrTable *table);**

FrVect: Vectors handling

FrVectNew

- This function create a multi dimension vector.
- Syntax: **struct FrVect *FrVectNew (int type, int nDim, ...);**
- The parameters (provided by the user) are:
    - type the type of data stored. It could be one of the following value:
      FR_VECT_C, /* vector of char */
      FR_VECT_2S, /* vector of signed short */
      FR_VECT_4S, /* vector of signed int */
      FR_VECT_8S, /* vector of signed long */
      FR_VECT_1U, /* vector of unsigned char */
      FR_VECT_2U, /* vector of unsigned short */
      FR_VECT_4U, /* vector of unsigned int */
      FR_VECT_8U, /* vector of unsigned long */
      FR_VECT_8R, /* vector of double */
      FR_VECT_4R, /* vector of float */
      FR_VECT_8C, /* vector of complex float (2 words per number)*/
      FR_VECT_16C, /* vector of complex double (2 words per number)*/
      FR_VECT_STRING; /* vector of string *
      FR_VECT_2U, /* vector of unsigned short */
      FR_VECT_8H, /* half complex vectors (float) (FFTW order) */ (not part of the frame format; convert to 8C when writing to file)
      FR_VECT_16H, /* half complex vectors (double) (FFTW order) */ (not part of the frame format; convert to 16C when writing to file)
    - nDim the number of dimension (1 for a vector, 3 for a “movie”: a set of images,...). If one of the dimension is the time, it must be the first dimension to be properly handled when chaning the duration of a frame.
    - nx[0] The number of element for each dimension (0 is a valid value)

    - dx[0] The step size for each dimension
    - unitX[0] The unit for each dimension .
    - Then, additional parameters for multi dimension vectors:
      nx[1], dx[1], unitX[1], nx[2],...
- This function return NULL in case of problem (not enough memory). After creation, all the different type of pointer in the FrVect structure point to the same data area. The names of these pointers are:

```
char *data;
short *dataS;
int *dataI;
long *dataL;
float *dataF;
double *dataD;
unsigned char *dataU;
unsigned short *dataUS;
unsigned int *dataUI;
unsigned long *dataUL;
```

- Remark: by default, the vector space is initialized to zero. To bypass this iinitialization, put a minus sign in front of the type argument.

FrVectNewTS

- This function creates a one dimension time serie vector. Like for an FrAdcData, the vector type is set according the number of bit (integer for positive values, float or double for negative value)
- Syntax: **FrVect *FrVectNewTS (char *name, double sampleRate, int nData, int nBits)**
- The parameters (provided by the user) are:

  name the name of the vector
  sampleRate: sampling frequency
  nData The number of elements (0 is a valid value)
  nBits: number of bits.

FrVectNew1D

- This function creates a one dimension vector.
- Syntax: **FrVect *FrVectNew1D (char *name, int type, int nData, double dx, char *unitX, char *unitY)**
- The parameters (provided by the user) are:

  name the name of the vector
  type the type of data stored (see FrVectNew).
  nData The number of elements (0 is a valid value)
  dx The step size
  unitX The step unit (NULL is a valid value) .
  unitY The content unit (NULL is a valid value) .

FrVectFree

- This function free a vector and its memory allocated space
- Syntax: **void FrVectFree (struct FrVect *vect)**

FrVectCompress

- This function compress a vector.
- Syntax: **void FrVectCompress (FrVect *vect, int compress, int gzipLvl)** were:

- vect is the vector provided by the user
- compresses the type of compression:
  - 6 for differentiation and zeros suppress for short and gzip for other
  - 7 for differentiation and zeros suppress for short, int and float to integer (not part of the current frame format)
  - 8 for differentiation and zeros suppress for short, int and float. (not part of the current frame format)
  - 255 for user defined compression code (definitely not part of the frame format)
- gzipLvl is the gzip compression level (provided by the user). 0 is the recommended value.

- In normal use, the compression is done at frame write and the user do not need to take care of it.

FrVectCopy

- This function duplicates a vector and its data:
- Syntax: **FrVect *FrVectCopy (FrVect *in)**
- This function returns NULL in case of problem (not enough memory).

FrVectCopyToF, FrVectCopyToD, FrVectCopyToI, FrVectCopyToS

- Syntax:
  FrVect * = FrVectCopyToF(FrVect *vect, double scale, char* newName);
  FrVect * = FrVectCopyToD(FrVect *vect, double scale, char* newName);
  FrVect * = FrVectCopyToI(FrVect *vect, double scale, char* newName);
  FrVect * = FrVectCopyToS(FrVect *vect, double scale, char* newName);
- These functions create a new vector of type float (FrVectCopyToF), double (FrVectCopyToD), int (FrVectCopyToI) or short (FrVectCopyToS). The data are copy using the scale factor 'scale' and casted to the proper type. The new vector will have the same name as the original one except if a newName is provided (value non NULL).
- These functions return NULL in case of error (malloc failed, no input vector).
- Supported input types: all types except complex. Syntax: **FrVect * = FrVectCopyTo(FrVect *vect, double scale, FrVect *copy);**
- This function copy the data from vector vect to the vector copy using the scale factor 'scale'and casted to the vector copy type.
- This function returns NULL in case of error (malloc failed, no input vectors).
- Supported types: all types for vect except complex: float, double, int and short for copy.<

FrVectDump

- To dump a vector in a readable format:
- Syntax: **void FrVectDump (FrVect *vect, FILE *fp, int debugLvl)** were
  - vect is the vector provided by the user
  - fp is the file pointer where the debug information will be send.
  - dbglvl is the debug level provided by the user. 0 means no output at all, 1 gives a minimal description (<5 lines per frame), 3 give you a full vector dump.
- Example: FrVectDump (vect, stdout, 1) will dump the vector information on the standard output.

FrVectDecimate

- This function decimates the vector data by averaging nGroup values together if nGroup is positive. If nGroup is negaive, a pure decimation (no averaging) of -nGroup is performed. The result is put in the vector outVect. (outVect could be the input vector). If outVect = NULL, the result is put in the input vector. The size of outVect should be nGroup time smaller than the size of vect.
- Syntax: **FrVect * FrVectDecimate (FrVect *vect, int nGroup, FrVect *outVect)**
- Examples:
  - FrVectDecimate (vect, 2, NULL) will average two by two the vector content of vect.(0, 1, 2, 3, 4, 5...) -> (0.5, 2.5, 4.5...)

  - FrVectDecimate (vect, -2, NULL) will take one values out of two input values.(0, 1, 2, 3, 4, 5...) -> (1, 3, 5...)

FrVectDecimateMin, FrVectDecimateMax

- These functions decimate the vector data by taking the minimum (maximum) values over nGroup values. The result is put in the input vector and the memory allocated is schrinked.
- Syntax: **FrVect * FrVectDecimateMin (FrVect *vect, int nGroup)**
- Syntax: **FrVect * FrVectDecimateMax (FrVect *vect, int nGroup)**
- Examples:
  - FrVectDecimateMin (vect, 2) will transform (0, 1, 2, 3, 4, 5) to (0, 2, 4)
  - FrVectDecimateMax (vect, 2) will transform (0, 1, 2, 3, 4, 5) to (1, 3, 5)

FrVectExpand

- This function uncompressed a vector:
- Syntax: **void FrVectExpand (FrVect *vect)** where vect is the vector provided by the user
- In normal use the uncompressed is done by a FrameRead call or by the channel access.

FrVectExtend

- Syntax: **void FrVectExtend (FrVect *vect, int nTimes, FrVect *outVect, char *newName)**
- This function extend the data from the vector vect by duplicate nTimes each values and returns the extended vector (outVect).
- The result is put in the vector outVect. The size of outVect should be nTime larger than the size of vect. values.
- If outVect is NULL, the output vector is created and named "newName" or as the original vector is newName = NULL.

FrVectFillX

- This function add one value at the end of the vector. The vector size is automatically increased.
- Syntax:
  - **int FrVectFillC (FrVect *vect, char value);**
  - **int FrVectFillD (FrVect *vect, double value);**
  - **int FrVectFillF (FrVect *vect, float value);**
  - **int FrVectFillI (FrVect *vect, int value);**
  - **int FrVectFillS (FrVect *vect, short value);**
- This function returns 0 in case of success or a non zero value in case of problem (not enough memory).

FrVectFindQ

- For a vector of string, this function returns the index of the string which match the parameter "name" or a negative value if not found.
- Syntax: **int FrVectFindQ (FrVect *vect, char *name)**

FrVectGetIndex

- Syntax: **FRLONG FrVectGetIndex(FrVect *vect, double x)**
- This function returns the bin index for a a given 'x' abcisse.
- It returns
  - -1 if vect == NULL
  - -2 if vect->dx[0] == 0
  - -3 if x is lower than the vector start (vect->startX[0])
  - -4 is x is larger than the vector end.

FrVectGetTotSize

- This function returns the total memory used by a FrVect structure (in bytes)
- Syntax: **FRLONG FrVectGetSize (FrVect *vect)**

FrVectGetValueI

- This function returns the bin content for a vector at index 'i' or zero if 'i' is outside the vector boundaries.
- Syntax: **double FrVectGetValueI (FrVect *vect, FRULONG i)**
- This function replace the obsolete function FrVectGetV.

FrVectGetValueGPS

- This function returns the bin content for a vector asusming that the x axis is GPS time. The overall GPS value is also subtracted.
- Syntax: **double FrVectGetValueGPS (FrVect *vect, double gps)**

FrVectGetValueX

- This function returns the bin content for a vector at position 'x' or zero if 'x' is outside the vector boundaries. The vector->startX is subtracted before getting the value.
- Syntax: **double FrVectGetValueX (FrVect *vect, double x)**

FrVectHtoC

- This function convert an half complex vector to a regular vector.
- Syntax: **int FrVectHtoC (FrVect *vect)**
- This function returns 0 in case of success or a non zero value in case of problem (not enough memory).

FrVectIsValid

- This function check that all floating points contained in a vector are valid IEEE floating point numbers.
- Syntax: **itn FrVectIsValid (FrVect *vect)** where vect is the vector provided by the user
- This function returns 0 if the vector does not contains NaN or INF numbers or if the vector contain only integers. It returns a non zero value (the index of the first bad value +1) in the other cases.
- remark: -0.(minus zero) is a valid floating point for FrVectIsValid.

FrVectLoad

- This function read from file a vector which has been saved with the function FrVectSave. It returns NULL in case of error.
- Syntax: **FrVect* FrVectLoad(char *fileName)**

FrVectMean

- This function return the vector mean value or 0 in case of problem.
- Syntax: **double FrVectMean(FrVect *vect)**

FrVectMinMax

- This function computes the min and max value of the input vector vect. It returns 1 in case of failure or 0 in case of success.
- Syntax: **int FrVectMinMax(FrVect *vect, double *min, double *max)**

FrVectRMS

- This function return the vector RMS value or -1 in case of problem.
- Syntax: **double FrVectRMS(FrVect *vect)**

FrVectResample

- Syntax: **FrVect *FrVectResample(FrVect *vect, int nDataNew, FrVect *outVect, char* newName)**
- This function resample the dData data from the vector vect to nDataNew values. It returns the resampled vector. The result is put in the vector outVect that must have the right size (but could have diffrent type). If outVect is NULL, the output vector is created and named "newName" or as the original vector is newName = NULL.

FrVectSave

- This function write on file a vector. It returns 0 in case of success. The vector could be read back using the FrVectLoad function.
- Syntax: **int FrVectSave(FrVect *vect, char *fileName)**
- If fileName == NULL, the output file name is "vectorName_vectorGPStime.vect"

FrVectSetName

- This function set or reset the vector name
- Syntax: **void FrVectSetName(FrVect *vect, char *name)**

FrVectSetUnitX

- This function set or reset the vector first dimension name
- Syntax: **void FrVectSetUnitX(FrVect *vect, char *unitX)**

FrVectSetUnitY

- This function set or reset the vector 'Y' dimension name (bin content)
- Syntax: **void FrVectSetUnitY(FrVect *vect, char *name)**

FrVectToAudio

- This function convert an vector to an audio file (format ".au" with 16 bits dynamic). The ".au" extension is added to the output filename. The sound is automatically adjust to use the full dynamic. The parameter "option" is unseed for the time being.
- Syntax: **void FrVectToAudio(FrVect *vect, char *fileName, char *option)**

FrVectZoomIn

- Syntax: **int FrVectZoomIn(FrVect *vect, double start, double length)**
- This function change the vector boundaries. After this call, the vector start at "start" and sas a length "length". The new vector boundaries could only be within the original boundaries. The unit used is the vector x unit.
- This function works only for one dimension vector.
- It returns 0 in case of success or an error code.

FrVectZoomInI

- Syntax: **int FrVectZoomInI(FrVect *vect, int iFirst, int nBin)**
- Same as FrVectZoomIn except that the arguments are bin numbers.

FrVectZoomOut

- Syntax: **int FrVectZoomOut(FrVect *vect)**
- This function cancel the effect of any previous FrVectZoomIn call. It returns 0 in case of success or an error code.

Frame Library Error Handling

Several errors may occurs during the code execution. A typical one is the failure of the memory allocation. In this case, the functions return NULL. But when the error occurs, a default handler is called. This handler is the following:

```
/*---------------------------------------------- FrErrorDefHandler---*/
void FrErrorDefHandler(level,lastMessage)
int level;
char *lastMessage;
/*-------------------------------------------------------------------*/
/* default handler for the FrameLib error. */
/* input parameters: */
/* lastMessage: the string which contain the last generated message */
/* level: 2 = warning, */
/* 3 = fatal error: requested action could not be completed*/
/*-------------------------------------------------------------------*/
{
if(FrDebugLvl > 0)
{fprintf(FrFOut,"%s",lastMessage);
fprintf(stderr,"%s",lastMessage);}
return;}
```

If the debug level (dbglvl) set by the call to FrLibIni has a value > 0 this handler print debug information on stderr and on the debug output file. This handler could be changed by the user at the initialization by calling the function:
void FrErrorSetHandler (void (*handler) (int, char *));
At any time the user can get the history of the errors (recorded in one string) by using the function:

char *FrError (0," ","get history");

The Frame Library Installation

Copyright and Licensing Agreement:

This is a reprint of the copyright and licensing agrement of the Frame Library:

> Frame Library Software Terms and Conditions
> This software is distributed under the terms of the GNU LGPL license
> Copyright 2018 Benoit Mours

Installing the library and associated tools

This software is available from https://git.ligo.org/virgo/virgoapp/Fr

Then use cmake, meson or the simple scripts:

- One script (makegcc) available in the mgr directory build the library (including a shared library). It uses the GNU (gcc) compiler. The binary is placed in a directory named by the system type (like SunOS or OSF1) in order to work in a multi platform environment. Remark: On Mac OS X you need to use the makeMacOS script instead of the standard makegcc script.
- To compile the examples/test program, use the script maketest, after using the script makegcc.
- To compile on Alpha, using the alpha compiler, use the script makealpha.

If you run on a non standard system, you may want to change the low level I/O function calls. By default, the Unix function call are used. To use the standard C FILE library you should compile the code using the option -DFRIOCFILE. To do more specific changes to the I/O you just need to change the FrIO.c file which group all those function call.

To use a user defined compression code (compression = 255) the functions FrVectUComp and FrVectUExpand should be provided by the user and the library should be compiled with the option -DFR_USER_COMPRESSION.

To use long long types you can compile the library with the -D FR_LONG_LONG option.

For any questions about this software, please contact Benoit Mours (benoit.mours@iphc.cnrs.fr).

Computer requirement for the Frame Library

The Frame software requests that the computer is at least a 32 bits computer. The Frame software writes the data in their original size and format. When reading the data on a different hardware, the frame library performed the byte swapping if needed (big-endian versus little-endian). It also expends or truncates the INT_8 variables if one machine has only 32 bits integer. The floating point variables are assumed to be always in IEEE format. The frame software (and installation scripts) has been tested on the following platforms:

- Alpha
- Linux
- Sun Solaris
- HP-UX
- Power PC under LynxOS
- Cygnus (GNU under Windows)

The Frame Library is ANSI-C code with POSIX compliance.

Test procedure

Once the library and the example have been installed, you can test it by running these examples. The prefix example has been replace by Fr. So to run the exampleFull, go in your machine sub directory and run FrFull. The first obvious thing to check is that the example run completely without crashing. Then some of the examples run in loop (like FrMark, FrMultiR, FrMultiW). They more designed to search for memory leak and it would be a good idea to check that the program size stay constant. Most of these tests created an frame file called test.dat. Each time this file is created, it is a good idea to run the utility FrDump with debug 1 and 2 and 3 on these file to check that the file content looks right. The suggested test sequence is:

- FrFull No arguments are needed. This test program produce a file called test.dat with different type of channel. Try "FrDump -i test.dat -d 3" to check if the file can be read.
- FrMark No arguments are needed. This program loop many time on the filecall test.dat. Check that the program size is stable (no memory leak)

- FrStat No arguments are needed. This test program produces a file (called test.dat) with static data. Check that you can read the file with FrDump.
- FrMark No arguments are needed. It will use the test.dat file produced with static data.
- FrOnline No arguments are needed. This program write frame in memory. It could be used to search for memory leaks
- FrMultiW No arguments are needed. This program creates 10 differents files which will be used by FrMultiR
- FrMultiR No arguments are needed. This program open and close many files. Usefull for memory leak.
- FrCompress No arguments are needed. This program test the compression algorithms. It should end with the message "Compression test OK"
- FrSpeed You should provide a file name and compress level. This program is used to estimate the reading/writeg speed.

The Matlab interface

Introduction

Matlab is a popular numeric computation and visualization Software. Since the Frame library is a plain C software, the connection between frame files and Matlab is easy to set. In the FrameLib package there is a matlab directory which contains:

- two MEX-file: frextract.c and frgetvect.c
- one script to compile the MEX-file: mymex
- three M-file to illustrate the use of the MEX-file:
  - exampleGetAdc.m Shows how to extract the Adc data from a frame file (using frextract), to plot a time series and it's FFT.
  - exampleGetVect.m Shows how to get a vector from a frame file with random access (using frgetvect), to plot a time series and it's FFT.
  - exampleAudio.m Shows how to produce and audio file from a frame file.

The purpose of this interface is to provide a direct path to extract data from a frame.

Matlab interface setting installation

The first operation to set the MEX-file is to compile it. This is done using the script mymex (just type ./mymex from the matlab directory).

Using frextract:

The frextract function could be called with the following arguments:

- Input arguments:
  1. file name(s). This could be a single file, a list of file separeted by space or a frame file list (ffl)

  1. ADC or PROCdata name (do not add extra space around the name)

  1. (optional) file index of the first frame used (default = first frame in the file)

  1. (optional) number of frame (default = 1 frames)
- Returned Matlab data:
  1. ADC or PROC data (time series)

1. (optional) x axis values relative to the first data point. This is usual time but it is frequency in the case of a frequency serie.
1. (optional) frequency values in the case of time series (double) (usefull for FFT's)
1. (optional) GPS starting time (in second.nanosec)
1. (optional) starting time as a string
1. (optional) ADC or PROC comment as a string
1. (optional) ADC or PROC unit as a string
1. (optional) additional information: it is a 9 words vector which content the variables: crate #, channel #, nBits, bias, slope, sampleRate, timeOffset(S.N), fShift, overRange (or the equivalent for proc data). All these values are stored as double

Using frgetvect or frgetvectN

The frgetvect function performs a random access in the frame file using the table of content. The function frgetvectN is doing the same think but the returned vector is normalized in the case of an ADC channel (using the “slope” parameter). This function is much faster than frextract when working with large file. This function could be called with the following arguments:

- Input arguments:

  1) file name(s). This could be a single file, a list of file separeted by space or a frame file list (ffl)

  2) channel name (it could be an ADC, SIM or PROC channel)

  3) (optional) starting GPS time(default= first frame in the file)

  4) (optional) vector length in second (default = 1 second)

  5) (optional) debug level (default=0; -1=no output; > 1 more out.))

- Returned Matlab data:

  1) ADC or SIM or PROC data stored as double

  2) (optional) x axis values relative to the first data point. This is usual time but it could be frequency in the case of a frequency serie (double)

  3) (optional) frequency values in the case of time series (double)

  4) (optional) GPS starting time (in second, stored in double)

  5) (optional) starting time as a string

  6) (optional) vector unitX as a string

  7) (optional) vector unitY as a string

All values are stored as double

Using other Frame tools with Matlab:

Do not forget also than you can run any Frame Utility program from Matlab by using the shell escape command ! For instance:

! FrDump -i ../data/test.dat

will call the program FrDump with the argument ran.dat.

Remark: if you call frgetvect and an exception is thrown (mexErrMsgIdAndTxt, etc), it is supposed to print its error message and dump you back to the commandline, but instead, it aborts, exiting matlab completely. To fix that, add "-fexceptions" to the build options in mgr/makegcc.

The ROOT interface

Introduction

ROOT is a powerful interactive environment developed at CERN (http://root.cern.ch). Among its various tools, It provide a very nice interactive C/C++ interpreter and detailed histograms capability. In the root directory of the Frame Library you will find a few scripts and macro to use the frames in the ROOT environment.

Frame library installation for ROOT

Assuming that you have already installed ROOT on your computer, you need first to build a special shared library. To do that, just adapt the build script to your system. You need at least to change the path to the ROOT directory and you may need to change some of the compilation flags... Once this is done, you need to update the PATH and LD_LIBRARY_PATH to include the FrameLib binary directory (named by your system). Then if you start root from the Fr root sub directory, it will execute the FrLogon.C which load everything you need.

Using the Frame Library in ROOT

Once ROOT is properly started, any Frame Library function is available as a ROOT command. Then 2 ROOT macro have been provided to build histograms out of the FrAdcData and the frame vector (FrVect). Just look at the three macro example to see what you can do. The FrVect vectors play a key role in these interactive analysis and more complex programs have been developed to provide direct interface to FFT and signal processing. See the Frv package (see http://wwwlapp.in2p3.fr/virgo/FrameL) and the Vega package (http://wwwlapp.in2p3.fr/virgo/vega). The test.dat file used by the exampleAscii.C and exampleAdc.C macros could be generated by running the FrFull example.

ROOT macros availabe:

- FrVP; This macros convert one or more vector to one histogram
  - TH1F* FrVP(FrVect *vect, char *draw = NULL, int color = 1, double xStart = 0., double xEnd = 0., double scale = 1.) This macros plot a single vector. Draw could take the value NULL to just build the histogram, "DRAW" to build and draw it or "SAME" to build and draw it on top of an existing histogram.
  - TH1F* FrVP(FrVect *vect1, FrVect *vect2 = NULL, double scale2 = 1., FrVect *vect3 = NULL, double scale3 = 1., FrVect *vect4 = NULL, double scale4 = 1.) This macros plot up to four vectors. The vectors 2 to 4 could be rescaled.
- FrAP: To plot and Adc channel:
  - TH1F* FrAP(FrAdcData *myadc, char *draw = NULL) This macro plot an ADC channel.

- FrCP: To plot a channel giving a file name and channel name(s):
    - TH1F* FrCP(char *fileName, double tStart = 0., double len = 2., char *channel1, char *channel2 = NULL, double scale2 = 1., char *channel3 = NULL, double scale3 = 1., char *channel4 = NULL, double scale4 = 1.)
    - TH1F* FrCP(char *fileName, char *channel1, char *channel2 = NULL, double scale2 = 1., char *channel3 = NULL, double scale3 = 1., char *channel4 = NULL, double scale4 = 1.)

The Octave interface

Introduction

GNU Octave www.octave.org is a high-level language, primarily intended for numerical computations. The interface frame to octave contains two routines [loadadc, loadproc] for loading ADC and PROC data from a given frame file into the Octave context. It has great similarities with the interface to Matlab previously described.

How it works?

Here is a description of what input variables should be provided to the loading interface and what output variables are available to the user:

LOADADC: Download an ADC signal in the Octave workspace from a given frame file.
Usage: [adc,fs,valid,t0,timegps,unit,slope,bias] = loadadc (fileName,[adcName[,nFrames[,first]]])
Input parameters:

- fileName: the name of the frame file
- adcName: [opt] the name of the ADC signal to be extr. [if missing: send a dump of fileName]
- nFrames: [opt] the number of frames to be extr. [if missing: send a dump of adcName frames]
- first: [opt] number of the first frame to be extr. [default=first frame avail.]

Output parameters:

- adc: the ADC signal
- fs: the sampling frequency
- valid: an index specifying whether the data are OK or not
- t0: the GPS time associated to the first bin in the first extracted frame
- timegps: [string] same thing but human readable format
- unit: physical units of the signal
- slope: slope coef. used to calibrate the signal X
- bias: bias coef. used to calibrate the signal X

LOADPROC: Download an PROC signal in the Octave workspace from a given frame file.
Usage: [proc,fs,t0,timegps] = loadproc (fileName,[procName[,nFrames[,first]]])
Input parameters:

- fileName: the name of the frame file
- procName: [opt] the name of the PROC signal to be extr. [if missing: send a dump of fileName]
- nFrames: [opt] the number of frames to be extr. [if missing: send a dump of procName frames]
- first: [opt] number of the first frame to be extr. [default=first frame avail.]

Output parameters:

- proc: the PROC signal
- fs: the sampling frequency
- t0: the GPS time associated to the first bin in the first extracted frame
- timegps: [string] same thing but human readable format

SAVEADC: Write an ADC signal from the Octave workspace to a given frame file.
Usage: status=saveadc(fileName,signalName,data[,fs,[t0]])
Input parameters:

- fileName: the name of the output frame file
- signalName: name of the ADC signal to be written
- data: input data (column vector of double)
- fs: sampling frequency
- t0: GPS time associated to the first bin of the first output frame

Output parameters:

- status: report about the writing operation.

SAVEPROC: Write an PROC signal from the Octave workspace to a given frame file.
Usage: status=saveproc(fileName,signalName,data[,fs,[t0]])
Input parameters:

- fileName: the name of the output frame file
- signalName: name of the PROC signal to be written
- data: input data (column vector of double)
- fs: sampling frequency
- t0: GPS time associated to the first bin of the first output frame

Output parameters:

- status: report about the writing operation.

Note that this description is also available online, by typing ``**help loadadc**'' or ``**help loadproc**'' at the octave prompt.

Test and getting started

A test script **plotframe.m i**s also part of the package. It uses the test framefile [test.dat] in the directory /data of the Frame Lib distribution. The script produces a plot of the first 1024 data points of the ADC signal 'Adc0', computes and plots its spectrum. This may be used as a start for learning how the interface works.

For any question about the Octave interface, please contact Eric Chassande-Mottin (ecm at obs-nice.fr)

The Python interface

This is the equivalent of the Matlab frgetvect interface.

To build the Python interface, look at the README file in the Python subdirectory.

Library Changes

For changes, see the git repo, or the SVN repo for previous change or in the documentation of v8r35 for earlier change description.