
Frame Library (Fr)
User's Manual

VIR-MAN-LAP-5400-103

Version 6.06

March 17, 2002

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-SPE-LAP-5400-102 and LIGO-T970130-E). 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 create 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.

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)>
-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)
-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)
  -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 are also checked if they are available. In case of success, this program returns a positive value, the number of frames in the file. It returns a negative value 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).
-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.
- 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.

Frame Handling

FrameCompress,	FrameMerge	FrameReshape
FrameCopy,	FrameRead,	FrameTagXXX,
FrameDump,	FrameReadN,	FrameUntagXXX,
FrameDumpToBuf	FrameReadRecycle	FrameWrite
FrameExpand,	FrameReadT	FrameWriteToBuf
FrameFree,	FrameReadTAdc,	FrameRemoveUntaggedData
FrameGetV	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

FrameGetV

- Syntax:
  FrVect* FrameGetV (frame, name)
- This function returns the pointer to the vector for one channel (Adc, Proc or FrSimData). Name is the channel name. It return NULL if the channel is not found.

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.

FrameReadRecycle

- This function read the next frame in a file. It returns NULL in case of error or end of file. If free the space for the previous frame and recycle it's static data. This function speeds up the read of files with lot of static data.
- Syntax: **FrameH* FrameReadRecycle (FrFile *iFile, FrameH *frame)** where iFile is the pointers to the input file provided by the user and frame a pointer to the frame to recycle or NULL.

FrameReadT

- This function read the frame starting at a given time. 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, 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.
  - name is a string containing a list of channel name separated by a space which could include wild card (example "*LSC* *EXC")

FrameReadTChnl

- This function is a driver for FrameReadTAdc, FrameReadTProc, FrameReadTSer, FrameReadTSim. It let the user to perfomer random access with a mixted type of channel. 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* FrameReadTChnl (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.
  - 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 move the information from the frame "frame" to the frame "new" which should be the output of the function FrameReshapeNew. This function perform 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 have 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

FrameTagXXX with XXX=Adc, Ser, Sim, Sum, Proc, Trig

- Syntax:

  void   FrameTag   (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   FrameTagSum (FrameH *frame, char *tag);
  void   FrameTagTrig (FrameH *frame, char *tag);

- These function 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 function. 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, Ser, Sim, Sum, Proc, Trig

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

FrameRemoveUntaggedData

- Syntax:
  void FrameRemoveUntaggedData (FrameH *frame);
- This function 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);**
- where:
  - frame is the pointer to the frame Header to be written
  - comp gives the compression algorithm used at writing time (see FrFileONew).
  - buf is the buffer which will receive the frame

    - nBytes is the buffer size
- This functions returns the number of bytes written or 0 in case of error.

FrAdcData: ADC's data manipulation

FrAdcDataDecimate

- This function reduce the sampling frequency of and ADC by averaging nGroup bins together (in this version, no filter is performed). It increase 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 functions returns 0 in case of success and a non zero value in case of error.

FrAdcDataDump

- This function produce 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 send (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 find 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);**

FrAdcDataGetV

- This function returns the vector (FrVect) for a given adc channel. It returns NULL it the the FrAdcData structure is not found.
- Syntax: **FrVect   *FrAdcDataGetV (FrameH *frame, char *adcName);**

FrAdcDataNew and FrAdcDataNewF

- These functions allocates 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 allocate 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);}
```

FrAdcDataReadT

- Syntax: **FrAdcData *FrAdcDataReadT (FrFile *iFile, char *name, double gtime);**
- This function perform 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 return 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 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.: **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 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.).

- 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).
  - 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].

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 channel for which we require some information (for instance, use tag = "*B1*" to get only the list of channel 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);**

FrFileIGetV:

- This function provide a 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 *FrFileIGetV (FrFile *iFile, char *name, double tStart, double length);**
- The returned vector start at the requested time and last exactly the requested number of second (this is new since version 5).
- 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 next field of the main vector. This vector gives for each expected frame 0 if the frame was not found or an integer value (usually 1) corresponding to the number of frame found for this time.
- After using it, the user should free the memory by calling FrVectFree since the FrVect structure has been directly extract from a file without frame to take care of memory clean up.

FrFileIGetVAdc, FrFileIGetSim, FrFileIGetProc:

- These functions are identical to FrFileIGetV, except that they search for only one kind of channel: FrAdcData, FrSimData or FrProcData.
- Syntax:
  - **FrVect *FrFileIGetVAdc (FrFile *iFile, char *name, double tStart, double length);**
  - **FrVect *FrFileIGetVSim (FrFile *iFile, char *name, double tStart, double length);**
  - **FrVect *FrFileIGetVProc (FrFile *iFile, char *name, double tStart, double length);**

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);**
- These functions extract 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 vector. The frame are sorted by increasing GPS time.

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 an input file for a given name..
- Syntax: **FrFile *FrFileINew (char *fileName);**
- This function returns a pointer to the input file or NULL if an error occurs.

- The string fileName could contain several file names. For instance the call to FrFileINew ("file1.dat file2.dat"); will open the file1.dat 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.
- Since version 4r40, random access works also on multiples files. Its is also possible to specify the start time and the length of each file to speed up the file access if many files are used simultaneously. Then these two numbers (file GPS starting time and file length in second) should follow the file name. For instance the following string could be used to work on three files, each of them being 1000 second long: "file1.F 657000000 1000 file2.F 657001000 1000 file3.F 657002000 1000"
- **Frame File List:** If there is only one file name and if the file extension is "ffl", then the FrFileINew function will read from this file the list of file to be processed. It is a very convenient way to provide efficient random access on a large number of files. The contain of the file is either a plain list of file or a list of file with GPS time and duration, has produced by the command "FrDump -i "your file list" -d 0". For instance, it could be:

  ```
  file1.dat    666000000.0000000    11.00
  file2.dat    666000011.0000000    11.00
  file3.dat    666000022.0000000    11.00
  file4.dat    666000033.0000000    11.00
  file5.dat    666000044.0000000    11.00
  ```

  For FFL, paths to files from the list are interpreted as absolute path.
- 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 compress.

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 *FrFileFewind (FrFile *file);**
- This function returns a pointer to the input file or NULL if an error occurs.

FrFileITFirstEvt, FrFileITLastEvt

- This function return 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 return 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 return 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 *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 which is string which will be added to the history record at writing time.
  - 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;

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 Checksum

- File checksum are computed when writing a file on disk. They are not computed/checked when the frame is write or if the frame is written in memory.
- To turn on(or off) the checksum computation/check just: iFile->chkSumFlag == FR_YES (or FR_NO); after the file has been open (FrFileIOpen or FrFileOOpen).
- The utility FrCheck could be used to verify the file checksum.
- Checksum are available only since version 4.40

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 add 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)**

- Find the associated vector in a Frame: FrVect ***FrProcDataGetV (FrameH *frameH, 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].

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")
  - 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.
  - File random access: **FrSerData *FrSerDataReadT (FrFile *iFile, char *name, double gtime)**

FrSimData

- These functions have the same meaning as for the AdcData structure:
  - Constructor: **FrSimData *FrSimDataNew (FrameH *frame, char *name, double sampleRate, int nData, int nBits);**
  - Dump: **void FrSimDataDump (FrSimData *simData, FILE *fp, int debugLvl);**
  - Destructor: **void FrSimDataFree (FrSimData *simData);**
  - Find it in a frame: FrSimData ***FrSimDataFind (FrameH *frame, char *name)**
  - Find the associated vector in a Frame: FrVect ***FrSimDataGetV (FrameH *frameH, 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:
    event = FrSimEventNew(frame, "CBSim","coalescing binariy", "2.5PN templates",
    710123123.44, 10., 0.1, 1.e-21, NULL, 2, "m1", 1.4, "m2", 1.4);
  - Parameters handling:
    - Add one more parameter to an event: **FrSimEvent *FrSimEventAddParam (FrSimEvent *event, char* paramName, double value);** This function returns NULL in case of error (bad input parameters of malloc failed).
    - 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].
  - 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 for more than one frame. It is written on tape 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 an 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.

FrStatDataAdd

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

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 find 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.

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 readt.
  FrStatData *FrStatDataReadT (FrFile *iFile, char *staticDataName, double gpsTime);

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, 2 for an image,...)
  - 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;
- For example to create a vector to hold image from a CCD camera (2 dimension vector of 512x512 pixels of 15 microns):
  vect = FrVectNew (FR_VECT_2S,2,512,15., "microns",512,15., "microns");
- 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)**

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).

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...)

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.

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.

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)**

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.

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)**

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.

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:

> Copyright (C) 2002, B. Mours.
>
> Frame Library Software Terms and Conditions
>
> The authors hereby grant permission to use, copy, and distribute this software and its documentation for any purpose, provided that existing copyright notices are retained in all copies and that this notice is included verbatim in any distributions. Additionally, the authors grant permission to modify this software and its documentation for any purpose, provided that such modifications are not distributed without the explicit consent of the authors and that existing copyright notices are retained in all copies. Users of the software are asked to feed back problems, benefits, and/or suggestions about the authors.
> Support for this software - fixing of bugs, incorporation of new features - is done on a best effort basis. All bug fixes and enhancements will be made available under the same terms and conditions as the original software,
>
> IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE

AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON-INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

Installing the library and associated tools

A compressed (gzip) tar file is available at http://wwwlapp.in2p3.fr/virgo/FrameL.

To uncompress it you should type:

```
gunzip vXrYY.tar.gz
tar xvf vXrYY.tar
```

Then you could either

- Simple script install:
  - 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.
  - 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.
- Using the configure GNU tools. To use the standard configure tools you just need to untar the various script and then you can run the usual told. In other words, the command you need to type are
  - cd vXrYY
  - tar xvf configure.tar
  - ./configure
  - make

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 (mours@lapp.in2p3.fr) or Didier Verkindt (verkindt@lapp.in2p3.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:
  - file name
  - ADC or PROCdata name (do not add extra space around the name)
  - (optional) file index of the first frame used (default = first frame in the file)
  - (optional) number of frame (default = 1 frames)
- Returned Matlab data:
  - ADC or PROC data (time series)
  - (optional) time values relative to the first data point
  - (optional) frequency values (for FFT's)
  - (optional) GPS starting time (in second.nanosec)
  - (optional) starting time as a string
  - (optional) ADC or PROC comment as a string
  - (optional) ADC or PROC unit as a string
  - (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:

The frgetvect function perform a random access in the frame file using the table of content (which of course need to be present). This function will be much faster than frextract when working with large file. This function could be called with the following arguments:

- Input arguments:
  - file name
  - channel name (it could be an Adc, Sim or Proc channel)
  - (optional) GPS starting time for the requested vector (default = first frame in the file)
  - (optional) vector length in second (default = 1 second)
- Returned Matlab data:
  - ADC data (time series)
  - (optional) time values relative to the first data point
  - (optional) frequency values (for FFT's)
  - (optional) GPS starting time (in second.nanosec)
  - (optional) starting time as a string
  - (optional) vector unitX as a string
  - (optional) vector unitY as a string

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.

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 system, 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)

Library Changes

From Version 2.37 to Version 3.10

- Several structures, structure's element names and vector types have changed. The new names follow the new Specification document. Some function names have been changed according these new names. The function names changes are:
  - - FrSmsxxx -> FrSerXXX
  - - FrRecXXX->FrProcXXX
  - - FrameDumpBuf -> FrameDumpToBuf
- In addition, the ASCII option for output file has been suppressed. The corresponding argument in the function FrOFileNew is now used for the frame data compression. Several static data with the same name but different time range can now be present in the same frame.

Files written with version 2.37 can be read by version 3.10.

From Version 3.10 to Version 3.20

- Add vector compression and fix bugs in FrVectCopy. Old files (from 2.3x) could still be read with the version 3.20.

From Version 3.20 to Version 3.30

- Change Utime to Gtime according to the specification LIGO-T970130-05.
- Add the variable Uleaps in the FrameH structure.
- Add the handling of FrSummary and FrTrigData structures.
- Fix bugs when writting compressed frames.

From Version 3.30 to Version 3.40

- Add the variable detector in the FrStatData structure.
- Change one parameter in the FrStatDataAdd function call (replace root by detector).
- Compute the number of bytes for the EndOfFile structure.
- Fix bugs in GPS time versus UTC time.
- Fix bugs in Floating point conversion with compression.

- Put I/O call in a separate file (there is one more file to compile so check your script).
- Improve the utility programs.

All files written with version 2.37 and higher can be read by version 3.40.

From Version 3.40 to Version 3.42

- Fix bug in FrFileIClose: the Static Data structures were not free.
- Improve the logic for static data update(data with timeEnd = 0 where not properly handled).

All files written with version 2.37 and higher can be read by version 3.42.

From Version 3.42 to Version 3.50

- Support multiple input file in FrFileINew.
- Remove some warning when reading old files
- Suppress the need to call FrLibIni
- Add the functions: FrameCopy, FrameHCopy, FrAdcDataCopy, FrameSelectAdc.
- Add a Matlab section.
- Update the examples

All files written with version 2.37 and higher can be read by version 3.50.

From Version 3.50 to Version 3.60 (March 22, 1998)

- Add the functions: FrDataFind.
- Fix the ADC sampleRate in exampleOnline.c and exampleMultiW.c
- Update the FrCopy FrDump and Frexpand utilities.
- Fix the bug in the directory creation in the makecc script.
- Fix bug in FrReadVQ (wrong malloc size).
- Add in situs framewrite

All files written with version 2.37 and higher can be read by version 3.60.

From Version 3.60 to Version 3.70 (Sep 16, 1998)

- Add the functions:
  - FrameWriteToBuf put a frame in one single buffer
  - FrameReadToBuf read a frame from a buffer
  - FrLibVersion return the FrameLib version number
- Fix bugs in:
  - FrameCopy : (uninitialized variable which in some case return the previous frame)
  - FrAdcDataNew : the 'adc' with floating point values where not properly created
  - FrVectWrite: write next vector if available
  - FrFile->Header fix the size from 32 to 40
  - FrReadLong and FrReadVL : logic for bytes swapping in the Pentium case.
  - FrameRead : in the case of reading unknown record
  - GPS time convention and associated print statement
  - FrSENew : the number of structure element of the dictionary was sometime wrong.
  - One variable name (localTime) in FrameH dictionary.
- Suppress the additional arguments in throvements in the case of Linux or DEC Alpha. See the FrIO.c file for details.
- Add a protection in FrVectNew if the function is called with strange arguments
- Add includes (unistd.h) in FrIO.h
- Change YES and NO global variables by FR_YES and FR_NO (internal variables)
- Add a parameters in the example FrDumpFile

- Print dictionary warning only if debugLevel > 1.
- Specify O_BINARY type for file open (needed for Windows NT)
- FrVectCompress: put a protection on gzipLevel (if set to 0 on Sun, the program crashed).
- Fix the format of a few print statements
- Update all the examples to use the latest functions and remove some unused variables

All files written with version 2.37 and higher can be read by version 3.70.

From Version 3.70 to Version 3.71 (October 6, 1998)

- Code cleaning in the FrCopy and FrDump utilities.
- Fix a bug in FrAdcDataNew when creating ADC?s with floating point values.
- Add a protection for bad arguments in FrAdcDataFind and FrSerDataFind in

All files written with version 2.37 and higher can be read by version 3.71.

From Version 3.71 to Version 3.72 (October 9, 1998)

- Use 315964811 to convert UTC to GPS time for old files instead of 315964810.

All files written with version 2.37 and higher can be read by version 3.72.

From Version 3.72 to Version 3.73 (November 11, 1998)

- FrVectCompress; add new compression scheme (zeros suppress for short) and try to optimize the code
- Add FrVectZComp and FrVectZExpand
- FrVectDiff: return the differentiate result (the original input is now unchanged)
- FrVectExpand: cleanup the logic
- FrVectWrite: To not copy the vector before compressing it
- FrVectDump: printf update

All files written with version 2.37 and higher can be read by version 3.73.

From Version 3.73 to Version 3.74 (April 2, 1999)

- FrAdcDataNew: Put adc name in the vector
- FrameDumpToBuf: Protect the case when the temporary file open failed.
- FrameWriteToBuf fix a bug to get the size including the EndOfFile record
- FrDicFree: Reset the file->SH pointer in order to be able to call directly FrDicFree
- FrFileINew and FrFileONew: Add protection for missing file name
- FrSimDataNew: Put data name in the vector and allow more type of storage
- FrSerDataGet fix bug to avoid name confusion with longer name
- FrProcDataNew: Change calling sequence to be compatible with Adc and Sim data.
- In FrVectCompress: compress only if stay in initial size and return the compress vector, the original is unchanged
- FrVectExpand: Trap error for unknown compression flag
- FrVectNew: do not fill the vector name
- FrVectNewTS: New function
- FrVectNew1D: New function
- FrVectWrite: do not compress if compress flag = -1
- FrIO.h: remove include to unistd.h
- exampleDumpFile.c: Add protection for missing file name
- frextarct.c: fix memory leaks, api description and printf statement for GPS starting time.
- Update the utilities FrDUmp.c FrCopy.c and FrExpand.c
- Clean up the FrameL.h to be compatible with ROOT/Vega

- Script: use only gcc and add the option -fPIC

All files written with version 2.37 and higher can be read by version 3.74.

From Version 3.73 to Version 3.75 (April 29, 1999)

- FrFileONew: restore the possibility to have fileName == NULL
- FrDicDump: fix a bug in the loop (this function is used only for debug)

All files written with version 2.37 and higher can be read by version 3.75.

From Version 3.75 to Version 3.80 (May 17, 1999)

- Change the FrIO.c code to support the standard C FILE.
- Add random frame access. This is an option which is under evaluation.

All files written with version 2.37 and higher can be read by version 3.80.

From Version 3.80 to Version 3.81 (June 4, 1999)

- Fix a bug in random frame access for multiple file open and close
- FrFileINew and FrFileONew: Removed protection for missing file name (added in 3.74)

All files written with version 2.37 and higher can be read by version 3.81.

From Version 3.81 to Version 3.82 (June 9, 1999)

- Add FrFileMarkFree to fix a memory leak when using the file marks.
- All files written with version 2.37 and higher can be read by version 3.82.

From Version 3.82 to Version 3.83 (June 15, 1999)

- Move the FrIO structure definition from FrIO.c to FrIO.h.

All files written with version 2.37 and higher can be read by version 3.83.

From Version 3.83 to Version 3.84 (August 23, 1999)

- FrVectWrite: Fix Memory leak when using compress.
- FrVectZComp: Add protection for buffer overflow

All files written with version 2.37 and higher can be read by version 3r84.

From Version 3.84 to Version 3.85 (September 11, 1999)

- FrTrigDataWrite and FrSummaryWrite: Fix bug to write link list
- struct FrVect: Add temporary variables (startX, frame, ?)
- Add define for GPS data and fix bug in time setting in the examples

All files written with version 2.37 and higher can be read by version 3r85.

From Version 3.85 to Version 4.00 (May 1, 2000)

- Change from frame format from version B to C. This generate many changes in the code. File written with FrameLib version 3.40 and above can still be read.
- Simplify the FrFileINew end FrFileONew API: the user do not need any more to pas a buffer. If he wants to directly write in a buffer he should use the FrameWriteToBuf function.
- Update the FrStatDataNew API to be in agreement with the new frame spec.
- Suppress the direct write in memory functionality

- Add miscellaneous feature, protections and fix various bugs:
- Add Gtime in FrVect
- Add the possibility to specify the history message produce at write time
- FrProcDataFind, SImDataFind: test if name == NULL
- FrAdcDataNew : remove the vect->name
- Fix bugs in compression flags logic.
- FrTrigDataWrite and FrSummaryWrite: Fix bug to write link list
- Reduce the use of long

All files written with version 3.40 and higher can be read with version 4.00

From Version 4.00 to Version 4.01 (May 15, 2000)

- Fix format of FrameL.h to be Root/Vega compatible.
- Add the functions: FrAdcDataNewF, FrAdcDataDecimate, FrameTagAdc, FrameUntagAdc, FrameMerge
- Remove the function FrameSelectAdc.
- Fix bug in FrameDumpToBuf.
- Change logic for static Data reading in order to keep them after a file close
- Fix a bug to be able to read file with format 4 after reading file with format 3

All files written with version 3.40 and higher can be read with version 4.01

From Version 4.01 to Version 4.02 (May 21, 2000)

- Fix various memory leaks.
- Add the functions: FrameTagSer, FrameUntagSer, FrVectStat

All files written with version 3.40 and higher can be read with version 4.02

From Version 4.02 to Version 4.03 (June 2, 2000)

- Fix two bugs for FrStatData which showed up for static data copy and multiple write in file.

All files written with version 3.40 and higher can be read with version 4.02

From Version 4.03 to Version 4.10 (October 11, 2000)

- Random access: Add a random access for a vector over several frames: FrameGetV. Add the function FrProcDataReadT, FrSerDataReadT, FrSimDataReadT. Add random access by run/frame number: function FrTOCFrameFindN. Change FrameReadT to work if not TOC available, Fix several bugs in random access. Add FrFileITStart(FrFile *iFile) FrFileITEnd (FrFile *iFile) functions. Add read function to frame Header FrameHReadN and FrameHReadT.
- improve debug for frame reading
- Define types for structures and reorganize FrameL.h
- Tag: Add Tag/untag function for Proc,Sim,Trig and Summary data. Add the function FrameTag, AntiTag is done by a word starting by -. Improve the wild cards for tag. Change the internal logic for Tag/Untag/find using a new structure (FrBasic). Protect Tag function for tag == NULL
- Add a function FrameNew (frameH with time + detectProc structures)
- Add a function FrFileONewH and FrFileONewFdH to define the program name in the history record.
- Add direct vector access: FrAdcDataGetV, FrProcDataGetV, FrSimDataGetV,
- Add a function FrVectGetV to access and convert a vector element.
- Table: Add FrTableExpand and the function FrTableGetCol. Fix a memory leak in FrTableFree
- To speed up frame creation, read and write: do not set to zero the vectors element when creating a new one.
- Update the examples and utilities to use the random access tools.
- FrMerge now do a full frame merge and not only the raw data.
- Fix bug in static data write (if on static data was changed it was not written on tape)
- Fix bugs for the FR_VECT_STRING vectors in several places (FrVectFree and New, Read, Write, Copy)

- Fix a memory leak in FrVectCompData
- Fix a bug in FrTrigDataNew: Copy timeBefore/timeAfter.
- Fix a bug in FrAdcDataNew to store nBit as an unsigned int.
- Fix the matlab interface.
- Fix a mismatch with the Frame spec for long: Write long as 8 bytes even on a 4 bytes computer (fix bug in ReadLong)

All files written with version 3.40 and higher can be read with version 4.10

From Version 4.10 to Version 4.11 (October 23, 2000)

- Change the way to write NULL string: now we write at least the '\0' character.
- Output files (oFile) and Table Of Content: add on option to not write the TOC for the time series (ADC, sim, ...).
- FrCopy.c : Add program name in history record, Add the option noTOCts to not write the TOC for the time series.
- Fix a bug in the zero supress compression algorithm in the case of consecutive values = -32768. (Update the test program exampleCompress.c)
- Fix bug in FrVectDump for vector of string with the string = NULL:
- Remove extra printf in FrLibVersion

All files written with version 3.40 and higher can be read with version 4.11

From Version 4.11 to Version 4.20 (December 7, 2000)

- Add buffer in FrIO. This speed up the disk read by a factor 2 to 3.
- Add new functions for random access: FrTrigDataReadT, FrSimEventReadT, FrameReadTAdc and check that all random access function skip FrSH and FrSE records.
- Add a new function FrameReadRecycle to speed up read of frame with lot of summary information.
- Update the following test program to test these new functions: exampleMark.c, exampleDumpFile.c
- Upgrade FrCopy to use the new random access tools and to allow it to uncompress already compressed files.
- Add decimation for float in FrAdcDataDecimate .
- Upgrade FrAdcDataReadT to work with wild card in the Adc name.
- Reorganize FrVectCompData to fix some bugs in the flags selection. (compression 5 was crashing for 4 bytes integers, could not uncompress a file already compressed).
- Add a test version of lossy compression for float.
- fix memory leak in FrSimEventXXX functions.
- fix a bug in FrSpeed in the speed computation.
- fix a bug in FrProcDataGetV.
- Add an Fr prefix to the zlib includes
- Fix some printf.

All files written with version 3.40 and higher can be read with version 4.20

From Version 4.20 to Version 4.21 (December 8, 2000)

- Fix a bug in FrAdcDataReadT (the wrong adc was read)
- Did some changes in the float to in compression.

All files written with version 3.40 and higher can be read with version 4.21

From Version 4.21 to Version 4.22 (December 12, 2000)

- Fix a bug in FrTOCFrameFindT which prevent to extract frame for files with only one frame.
- Add the convention for random access that if the requested time is 0, you get the first frame in the file.

All files written with version 3.40 and higher can be read with version 4.22

From Version 4.22 to Version 4.23 (Jan 16, 2001)

- Fix bug in FrFileIGetV to get the proper frame is TStart = 0.
- Fix a printf bug in FrLibVersion.
- Fix memory leak when using table of content with several file open/close (functions FrTOCevtRead and FrTOCtsRead)
- Add a protection in FrVectCopy if vectIn = NULL
- Add a protection in FrVectStat to prevent crash if the data are invalid.
- Add new function (FrVectIsValid) to check if a vector contains NAN or INF data.
- Add a random access matlab interface (frgetvect.c).
- Add an include in FrIO.c to fix compilation warning on Alpha with gcc.
- Update the ROOT build script to work also on Alpha. Change the name of the ROOT shared library.
- Change the frequency of one example FrFull .

All files written with version 3.40 and higher can be read with version 4.23

From Version 4.23 to Version 4.30 (Feb. 08, 2001)

- Full rewrite of the pointer relocation logic to speed up read/write for frames with many channels.
- Sort by alphabetic order the channels in the table of content.
- Fix a memory leak which was observed when using random access with table of content on many files.
- Add scripts for autoconf GNU tools (thanks to Duncan Brown).

All files written with version 3.40 and higher can be read with version 4.30

From Version 4.30 to Version 4.31 (Feb. 27, 2001)

- Fix bugs in the handling of static data. The static data were not properly read/write since version 4.30.
- Update the example exampleSpeed.c to compute more things.
- Update the root macro to add more option for the plots of one vector

All files written with version 3.40 and higher can be read with version 4.31

From Version 4.31 to Version 4.40 (July 11, 2001)

Thanks to Isidoro Ferrante, Martin Hewitson, Julien Ramonet, Andrei Sazonov, Peter Shawhan, Didier Verkindt and Andrea Vicere for finding and reporting bugs.

- Major rewrite of random access functions (most of them). Some new random access function added. Now random access is supported when using multiple files with usually wild card as well. The random access work properly now in case of missing frames. The definition of the search time window for FrTrigDataReadT as been update to be consistent with the other random access function. Fix a weakness in the random access. The GPS time is defined as double and sometime if we use the integer starting time converted to double we got the previous frame due to round off error.
- Add Frame File List option for the FrFileIOpen for efficient random access in multiple files.
- Add one more parameter to the functions FrTrigDataFind, FrSerDataFind, FrSimEventFind to be able to access structure with the same name in the same frame. Warning: code using these functions need to add this extra parameter which should be set to NULL to work as previously.
- Change the function FrAdcDataFree to free now the full linked list
- Improving some dump, debug and printout statements, especially in FrAdcDataDump, FrVectDump (with large debug level, you could have a full vector dump), and FrVectStat (add management of doubles) and a wrong comment about compression in FrCopy. Improve the table of content dump: add the number of frames containing the channel and allow the use of tag to control the channel list. To do that, the function to tag data have been reorganized.
- Add a new FrVect type: FR_VECT_H8 and FR_VECT_H16 for FFTW half complex vectors
- Add the type FR_VECT_8C as specify in the frame spec. The miss typed FR_VECT_C8 type is still valid.
- Fix a bug in FrTOCtsMark: The Adc group ID and channel ID were swapped. This means that all files written with previous version of the library have this information swap in the table of content. If the file is copy, the table of content is rebuild with the right information.
- In case of unknown compression flag, free the working space in FrVectExpand to avoid memory leak.
- Add the frame reshaping function (FrameReshapeXXX) to change the frame length.
- Add more feature to FrCopy like decimation and change of the frame length.
- Update FrAdcDataDecimate and FrVectIsValid to handle double precision floating points and to fix a bug in the update of nx[0].
- Add a test when writing data to check for duplicate channels.

- Fix a bug in FrFileORealloc (the memory allocation failure was not properly trapped).
- Fix bug in TOC writing on PC. The TOC of files writen on PC with version < v4r40 are not usable. To regenerate the TOC, just copy the file with the latest FrCopy utility.
- Add zero suppress compression options for 4 bytes integers and floating points numbers
- Add the prefix "Fr" to all gzip functions to avoid name conflicts.
- Add the function FrVectDecimate and FrStrUTC
- Add the computation of checksum when writing file.
- Update some example programs.
- Update the compilation script to work on cygnus also. The makecc script has been renamed makegcc to be more consistent and do not compile anymore the example. To compile them use the script maketest.

All files written with version 3.40 and higher can be read with version 4.40

From Version 4.40 to Version 4.41 (July 31, 2001)

Thanks to Sam Finn, Frederique Marion and Peter Shawhan for finding and reporting problems and bugs.

- Fix the checksum computation on SGI (FrChkSum function).
- Fix a bug in FrCopy when using input list from standard input (STDINLIST option)
- Fix FrADCDataReadT, FrProcDataReadT and FrSimDataReadT to avoid memory leak if there is no frame for the requested time.
- Fix bug in FrameReshapeNew to properly handle the case with non zero position.
- FrFileIGetVType: increase round off margin to deal with vector with a slight offset compared to the frame start time.
- Minor update of the TOC dump
- Initialize to zero any new vector, except the one created by FrAdcDataNew. (This initialization was suppress at version v4r10).
- Add the function FrVectMinMax.

All files written with version 3.40 and higher can be read with version 4.41

From Version 4.41 to Version 4.42 (Sept 19, 2001)

- Remove the to the FrCHkSum function (declare the first argument of the FrChkSum function as char * instead of signed char *) to be able to work with the current version of root in order to bypass a bug in CINT.
- Change FrVectIsValid to flag sub normal floating point numbers as invalid floating points.

All files written with version 3.40 and higher can be read with version 4.42

From Version 4.42 to Version 4.43 (Oct 15, 2001)

- Fix the FrVectIsValid function to not mark zero floating point value as invalid floting point.
- Fix a Bug in FrTOCSetPos to be able to do random access on file larger than 2 GBytes.

All files written with version 3.40 and higher can be read with version 4.43

From Version 4.43 to Version 4.44 (Nov 14, 2001)

- Fix a Bug in the zero suppress compression algorithm for floating point (compression flag = 8).

All files written with version 3.40 and higher can be read with version 4.44

From Version 4.44 to Version 4.50 (March 13, 2002)

Thanks to Damir Buskulic, Eric Chassande-Mottin, Ed Daw, Martin Hewitson, Frederique Marion Alain Masserot, Peter Shawhan and Didier Verkindt for suggestions, finding and reporting problems and bugs.

- FrVectIsValid: rewrite the code to fix a bug when checking double and to not trig on -0.
- FrVectDecimate: update the value of out->size = in->size/nGroup.
- FrVectStat: minor changes in the output format (replace 4 by %5).
- FrVectDump: for complex numbers: add protection for invalid complex number and provide full dump. Dump also GPS time if available.
- FrMerge: if no info timing for one frame, use the values from the other one.

- Fix a memory leak when using random access on multiples files (avoid the double reading of FrSH structures).
- FrTagMatch: Fix a bug in the case of 'multiples' antitag. For instance if the tag was "-adc1 -adc2", only "-adc1" was taken into account.
- Add a protection when performing random acces on file with channels missing for a few frames.
- FrChkSum: Select char or signed char according to the compiler definition.
- FrCopy: Uncompress data if the option decimate is used. Check that the option -a is requested after the input file declaration.
- FrVectRead/Write fix a bug to fullfil the frame spec in the case of uncompressed vector on littleendian machine. Warning: due to this change, uncompress data produced on littleendian machines will be unreadable with previous library version.
- FrFileINext and FrDum: add a protection on invalid file when reading multiples files.
- Add function FrMsgDump, FrameReadTSer, FrameReadTSim, FrameReadTProc, FrameReadTChnl.
- Add an Octavia directory (code from Eric Chassande-Mottin).
- Fix names in the gnu install scripts.

All files written with version 3.40 and higher can be read with version 4.50

From Version 4.50 to Version 4.51 (March 18, 2002)

Thanks to Andrei Sazonov, Peter Shawhan and Didier Verkindt for suggestions, finding and reporting problems and bugs.

- FrVectWrite fix a bug to fullfil the frame spec in the case of uncompressed vector on littleendian machine. Warning: due to this change, uncompress data produced on littleendian machines will be unreadable with previous library version.
- Add the function FrFileIChannelList.
- For FFL, paths to files from the list are now interpreted as relative to the path of the list file, not relative to the current directory.

From Version 4.51 to Version 4.52 (March 20, 2002)

- FrVectDecimate: Free the unused space at the end
- Fix a format in FrFileIChannelList.

All files written with version 3.40 and higher can be read with version 4.52

From Version 4.52 to Version 4.53 (May 2, 2002)

Thanks to Damir Buskulic, Jean-Marie Teuler and Didier Verkindt for suggestions, finding and reporting problems and bugs.

- FrReshapeAdd: Do not add images, just move them
- FrVectIsValid: Fix a bug introduced since v4r50 (in case of error, it returns now the index of the first invalid vector element)
- FrIO.h: add a protection for multiple includes.
- Add channel list (FrCList) functions for AdcData and SerData
- FrAdcDataDecimate: use a double for local sum (to avoid problem foir large float values)
- FrCopy: Use FrameReadTChannel instead of FrameReadTAdc
- Add the functions FrTOCFrameFindNext and FrTOCFrameFindNextT
- Fix a memory leak in FrExtract

All files written with version 3.40 and higher can be read with version 4.53

From Version 4.53 to Version 5.00 (Augst 12, 2002)

Many changes have been made, most of them to support the frame format version 5.

A few function calls have been changed, basically to add extra parameters.

- FrSerDataNew: add the sampleRate parameter.
- Rename FrTOCFrameFindNextT to FrFileITNextFrame.
- Rename FrTrigData to FrEvent.
- Change the calling sequence of FrEventNew and FrSimEventNew (GtimeN,S have been replace by a double and event parameters could be directly added).

- FrEventReadT and FrSimEventReadT : add two extra parameters to select event on the amplitude if needed.
- Change the return argument for FrHistoryFree and FrStatDataFree to be like the other Free function.
- FrFileIGetV return now a vector with the requested boundaries.

New functions have been added, especially to provide Table Of Content data access:

- FrFileIGetXXXNames and GetXXXInfo provide information from the TOC.
- FrFiltITFirstEvt and FrFileITLastEvt provide the time of the first and last event in the file.
- FrMsgFind find a FrMsg in a frame.
- FrVectNewL let you create vector with a long long argument type for nData.
- FrameGetLocalTime return the localTime offset for on Adc channel.
- FrProcDataAddHistory let you add history to an FrProcData channel.
- FrVectZoomIn and FrVectZoomOut to change a vector boundaries

New feature have been added:

- When doing random access check that we read the right vector.
- The name of the history record is now the frame name.
- Split the time boundary for frame and for events in the Frame File List.
- Add file statistic in FrDump (the debug flag level have slightly changed).
- Change FrIO.h for HPSS include when needed.
- FrCopy do not change the compression type by default now.
- FrAdcDataDecimate: realloc space at the end to free unused space.
- FrFileIOpen: add a protection for multiple calls on a file already closed
- FrFileIChannelList: print also the min and max value for each channel.
- upgrade FrCheck to check sequentiel reads and check individual frame checksums.

Several bugs have been fixed:

- The type and users vector in FrameH are now read, written to file and free when they are used.
- Fix a bug in FrVectDecimate (wrong realloc).
- In case of random access with some channels that are not in all the frames, the following channels were not read.
- FrPutLong was not properly working on some computers.
- FrFileINew: fix a bug when the ffl file was not present.
- FrCopy: when the requested starting time was after the end on file, all the file were copied.
- Complete the zlib prefix list in Frzconf.h.
- Add a protection when doing frameReshape on compressed frames.

Your compiler should flag all the change you need to do on your software to convert it to version 5. The most likely changes will to replace FrTrigData by FrEvent, update the new and find functions, remove the use of localTime from the frame header and fix some printf when Files written with version 3.40 and higher can be read with version 5.00

From Version 5.00 to Version 6.00 (Augst 14, 2002)

Since some LIGO software (the version of FrameCPP used by LDAS) produced frame labelled as version 5, but based an a intermediate draft version of version 5, it has been decided to relabelled the frame spec to version 6. Files written with FrameLib version 3.40 and higher can be read with version 5.00. The ‘version 5’ frames produced by FrameCPP could also be read by this new version of FrameLib, but only in a sequential mode (no random access using the TOC). The changes made between v5.00 and v6.00 are due to this modification.

From Version 6.00 to Version 6.01 (September 9, 2002)

Thanks to Damir Buskulic, Eric Chassande-Mottin, and Didier Verkindt for suggestions, finding and reporting problems and bugs.

- Fix a bug which prevent to performed random access read with FrEvent for version4 frames.
- FrFileIChannelList: fix a bug in the min/max values.
- Add functions : FrameRemoveUntaggedData and FrEventAddParam
- Describes Virgo dataValid values in FrAdcDataDump

- FrameReshape: protect it when using tagged frames and set dataValid flag for ADC if part of the ADC is missing
- Update OCTAVE interface:
    - Bug fix in loadadc/loadproc
    - New functions saveadc/saveproc
    - Update Makefile

Most files written with version 3.40 and higher can be read with version 6.01

From Version 6.01 to Version 6.02 (October 22, 2002)

Thanks to Damir Buskulic, Eric Chassande-Mottin, Frederique Marion and Soumya Mohanty for suggestions, finding and reporting problems and bugs.

- Add a protection agains NULL vector in FrVectCompress and FrVectExpandF
- Change FrameH->run from unsigned int to signed int.
- FrVectCompData: fix a bug for gzip compression: the output spce was slightly to short and in a very few cases, this was creating problems.
- Fix dictionnary for FrProcData->history, and for arrays in FrTOC (INT_4U * changed to *INT_4U for instance).
- Commented out the test compression method 7 (not part of the spec).
- Put the write nBytes value when writting vectors of string.
- FrEventNew and FrSimEventNew: add a protection for the case nParam=0 which generate some problems on Alpha.
- Increment FrSH and FrSE instance numbers.
- Add the function FrEventGetParam, FrSimEventGetParam and FrSimEventAddParam
- For the FrEventReadT function (and similar random function) read the full linked list of associated vector instead of just the first one.
- Set GTimeS for the vector assiocated to an event when performing a random access using FrEventReadT (and similar functions).
- Update the Matlab interface frextract to work also with FrProcData
- Update the data test files (and the exampleFull.c program to produce more channels)
- Remove the FrProcData->sampleRate variable.

Most files written with version 3.40 and higher can be read with version 6.02

From Version 6.02 to Version 6.03 (December 20, 2002)

Thanks to Damir Buskulic, Frederique Marion, Bernd Machenschalk, Jean-Marie Teuler, Gabriele Vedovato and Didier Verkindt for suggestions, finding and reporting problems and bugs.

- Add paramaters handling functions for FrProcData (FrProcDataAddParam,...)
- Fix a bug in FrProcDataRead: the GTimeN field was not properly used when computing the GPS time.
- FrVectMinMax: add a protection against invalid floating point numbers
- Read unsigned short instead of signed short for the length of 'char'. It provides the full length for 'char' and prevents segmentation fault when reading unvalid frames. This is transparent for regular frames.
- Add various consitency checks to protect against corrupted files when reading them. This allows to recovert some frame file version 4 with when doing random access on some missing channel.
- Fix a bug in FrFileIGetAdcNames to prevent to read the TOC for most of the files and when reading data randomly
- FrCopy: fix a bug in a malloc which was generating crashes on Linux
- Minor change in FrVectDump to avoid the access of missing vector element for short vectors and to print the position of the maximum value.
- Include the licensing agreement.

Most files written with version 3.40 and higher can be read with version 6.03

From Version 6.03 to Version 6.04 (February 11, 2002)

Thanks to Elena Cuoco, Bernd Machenschalk, Frederique Marion, Szabolcs Marka, Ed Maros, Michele Punturo, Peter Shawhan, Gabriele Vedovato, Didier Verkindt and John Whelan for suggestions, finding and reporting problems and bugs.

- Fix a bug introduced in v6r03 which prevent to access FrProcData in random access on some files.
- Fix a bug in FrVectZCompI (failing to compress a vector if the 4 bytes vector includes the value -1)
- Fix a bug in FrVectCompData: With the compression type 8, If a vector could not be differentiate, the vector type was corrupted.
- Fix a bug in the checksum computation on Sun since v6r00 (in FrEndOfFileWrite).
- In FrChannelList: Do not compute min/max. This fix a memory leak.
- FrVectDump: improve the full debug for a few rare cases (like 8U)
- Minor debug messages fixes: (FrEndOfFileRead: do not print checksum if it is not computed, FrFileIOpen,...)
- Protect FrIO.h against multiple includes
- Add setting function for FrAdcData (FrAdcDataSetAux, ...DataValid, ...SetFShift, ...SetTOffset)
- Add the function FrStatDataReadT (for random access) and FrStatDataDump.
- Upgrade FrVectDecimate to provide a true decimation (without averaging) if nGroup < 0.
- Upgrade the event handling:
  - FrEventNew and FrSimEventNew: add a warning for the multiple paramters type (they should be double).
  - Add the FrameAddEvent, FrEventCopy, FrEventReadData and FrEventReadTF functions and the equivalent function for FrSimEvent.

Most files written with version 3.40 and higher can be read with version 6.04

From Version 6.04 to Version 6.05 (February 25, 2002)

Thanks to Bernd Machenschalk, Frederique Marion and Didier Verkindt for suggestions, finding and reporting problems and bugs.

- Fix a bug in FrameAddEvent (when adding a linked list of events, only the first one was added, the other were lost).
- Fix a bug in FrChannelList: (in case of FrProcData, their was a segmentation fault).
- Fix a bug in FrFileIGetV (for too large vectors, a failed malloc was not protected and was producing a segmentation fault)

Most files written with version 3.40 and higher can be read with version 6.05

From Version 6.05 to Version 6.06 (March 17, 2002)

Thanks to Franco Carbognani, Frederique Marion, Ed Maros and Gabriele Vedovato for suggestions, finding and reporting problems and bugs.

- Change FrameAddEvent to add the events in direct order instead of reverse order in the linked list.
- Make shure that bytes 18 to 25 of the file header are properly filed on all machines.
- Fix a bug in FrDump (it was crashing on some platforms like Linux when more that one file was given as input argument).
- Small changes the build and environment scripts for root

Most files written with version 3.40 and higher can be read with version 6.06