Tools for Handling GMRT Native Data Recording Format

Sanjay Bhatnagar
N.C.R.A., Pune

March, 1997

1 Introduction

This report describes two C++ objects called `LTAFMT` and `LTAVIEW` which encapsulates the GMRT native data recording format called the *LTA Format*. This also describes the interface layers of the underlying C library. Unless absolutely necessary, programmers are encouraged to use the C++ objects for programming. The lowest layer of the C library is not described here.

Appendix A (see section 4) describes a library (called `glib`) which has functions of general use. This includes routines of general use as well as higher level routines for filling the antenna co-ordinate table (`struct AntCoord`) and the `fftmac` structure (functions `LoadAntTab` and `getFFTMac`).

The `LTAFMT (see section 2.1)` object described below is the base-class for handling LTA formatted files. This object encapsulates the functionality which is expected to be a common factor in all classes which will handle the LTA formatted files and do higher level operations (for example see section 2.2). All such higher level object (also called "derived" classes), inherited from this base-class, would have all the functionality of `LTAFMT (see section 2.1)` object plus the additional functionality of the inherited class.

`LTAVIEW (see section 2.2)` is an object derived from `LTAFMT`. Hence, all the methods of `LTAFMT` plus the methods described below are available from `LTAVIEW`. Application developers should work with `LTAVIEW` object (and *not* with `LTAFMT`).

`LTAVIEW` is designed to give a "view" of the data with selection on the source name and integration in time applied to the data. Hence, once the user supplies a source name, this object will behave exactly as `LTAFMT` except that it will give a view of the data base which has scans only of the selected source (source name can be a POSIX EGREP complaint regular express with the exception that "+" regular expression operator has to be "escaped").

The underlying C library called `hlib` (section 3) does most of the work. It starts by first checking if the given data base is in the LTA format, and then loads the global header in the memory and keeps it as a doubly linked list. This is done in the call `BeginHeader()` and leaves the file pointer at the first scan header. The header is available for the full life of the application (unless explicitly destroyed).

From this point onwards, the programmer can use the various functions of the library to step through the file, each time reading a full record. The library returns the type of the record read via the return value of the `rd_data`. If the library detects a scan header, it loads it in the memory as a doubly linked list. Only the latest scan header is maintained.

The global and scan headers can be modified using the library functions.

When writing the data to the output stream, the programmer must explicitly write the headers. `wt_data` would write the data to the output stream while `gwrthdr()` and `swrthdr()` methods would write the global and scan headers respectively.

2 C++ Object

2.1 The `LTAFMT` Object

This section describes the base class for performing I/O in LTA format. The class provides all the functionality provided by the lower level C library called `hlib` (section 3). Using this, one can open any number of LTA files in an application without any clash.

1. **LTAFMT()**

   The default constructor for the object. This is automatically called when the object is instantiated.

   Here, it simply initializes a blank object (with no data base file attached yet).

2. **LTAFMT(char *InFile, char *OutFile=NULL)**

   An overloaded constructor. This is called if the object is instantiated with the data base file name as an argument. The second argument is optional, and if given, should be the file name where the output data base would be written by this object.

3. **LTAFMT()**

   The default destructor. This is automatically called when the object goes out of scope or if the `delete` operator is applied on the object.

4. **virtual int reset(char *InFile, char *OutFile=NULL)**

   Method to reset the internals of the object. This is typically called if a blank object instantiated elsewhere in the code is to be reset (though this can be called for a non-blank object as well).

5. **FILE *getInpfd()**

   Returns the input file descriptor used by the object (breaking classical object oriented design!).

6. **FILE *getOutfd()**

   Returns the output file descriptor used by the object (breaking classical object oriented design!).

7. **int append(LTAFMT& DB)**

   Method to append the LTA data base object `DB` to the current object.

8. **virtual int getnscans()**

   Returns the number of scans in the data base detected by this object.

9. **virtual int skipscan(int n)**

   Method to jump to the scan number given by `n`.

10. **virtual int skiprec(int n)**

    Method to skip `n` records starting from the current record.

11. **virtual int rd_data(char *buf)**

    Method to fill `buf` with the next record in the data base. The method return `NEWSCAN` if the record was a new scan header, `DATAREC` if the record was a data record, `EOF` if the end-of-file was detected in the input, and `UNKNOWNREC` for any other type of record.

12. **virtual int rd_data()**

    Method to fill the internal data buffer with the next record in the data base. Return values are same as above.

13. **virtual int wt_data(char *buf)**

    Method to write the buffer `buf` to the output stream.

14. **virtual int wt_data()**

    Method to write the internal data buffer to the output stream.

15. **char *getDataBuf()**

    Return the pointer to the internal data buffer (breaking the classical object oriented design!).

16. **int getNFFT()**

    Returns the number of FFT pipelines detected in the database.

17. **int getNoOfBase()**

    Returns the number of baselines detected in the data base.

18. **int getNoOfChans()**

    Returns the number of frequency channels detected in the data base.

19. **void getChanList(intVec &chanlist) const**

    Returns the list of valid frequency channels in `chanlist`. This also resizes the `chanlist` object to the required size. The size of the list can be found using the `intVec::capacity()` method.

20. **void getBaseList(intVec &baselist) const**

    Returns the list of valid frequency channels in `baselist`. This also resizes the `baselist` object to the required size. The size of the list can be found using the `intVec::capacity()` method.

21. **int getRecLen()**

    Returns record length for the data base.

22. **void getSelfs(int b, int *s1, int *s1)**

    Returns the two MAC numbers in `s1` and `s2` (equivalently the baseline number) on which the self-correlations of the two antenna making the baseline `b` appear.

23. **int getFFTMac(struct fftmac *fm)**

    Fills `fm` from the data base (see the section on `FFTMAC` object below for details about the structure itself).

24. **int getAntTab(struct AntCoord *Tab)**

    Fills `Tab` with the antenna co-ordinate table read from the data base.

25. **double getTimeStamp(char *data=NULL)**

    Returns the time stamp of the current data buffer `data` in units of seconds. If no argument is supplied, it uses the internal data buffer to read the time stamp.

    If the buffer is not a data buffer, invalid time will be returned.

26. **void getHMS(int t,float *hms)**

    Converts the time in `t` (in units of seconds) to hour,minutes and seconds and returns in `hms`.

27. **int operator==(LTAFMT &)**

    Checks if the two `LTAFMT` objects are equivalent. Two objects are treated "equivalent" if the values of the following keywords match: `RECL, DATASIZE, DATA_OFF, PAR_OFF, BASELINE, CHANNELS, LTA, INTEG, BASE_NUM, CHAN_NUM`.

28. **void gwrthdr(FILE* fd=NULL)**

    Writes the global header to the file descriptor `fd`. If no argument or a NULL argument is given, the output goes to the output stream of the object.

29. **int ggetval(char *Key, float *Val)**

    Return the value of the keyword `Key` in `Val` from the global header as a floating point number.

    If the return value is -1, the keyword was not found.

30. **int ggetval(char *Key, double *Val)**

    Return the value of the keyword `Key` in `Val` from the global header as a double precision number.

    If the return value is -1, the keyword was not found.

31. **int ggetval(char *Key, int *Val)**

    Return the value of the keyword `Key` in `Val` from the global header as an integer.

    If the return value is -1, the keyword was not found.

32. **int ggetval(char *Key, char *Val)**

    Return the value of the keyword `Key` in `Val` from the global header as a string. `Val` should be at least of the size `LINELEN`.

    If the return value is -1, the keyword was not found.

33. **int gputval(char *Key, float Val)**

    Put `Val` as the value of the key word `Key` in the global header as a floating point number.

    If `Key` is already present in the header, it's value will be replaced, else the key will be added.

    If return value is -1, the operation failed.

34. **int gputval(char *Key, double Val)**

    Put `Val` as the value of the key word `Key` in the global header as a double precision value.

    If `Key` is already present in the header, it's value will be replaced, else the key will be added.

    If return value is -1, the operation failed.

35. **int gputval(char *Key, int Val)**

    Put `Val` as the value of the key word `Key` in the global header as an integer.

    If `Key` is already present in the header, it's value will be replaced, else the key will be added.

    If return value is -1, the operation failed.

36. **int gputval(char *Key, char *Val)**

    Put `Val` as the value of the key word `Key` in the global header as a string.

    If `Key` is already present in the header, it's value will be replaced, else the key will be added.

    If return value is -1, the operation failed.

37. **int gputhistory(char *Record)**

    Puts a history entry in the global header. If the `Record` is longer that 79 characters, the history entry would be split in more than one entry.

38. **void swrthdr(FILE* fd=NULL)**

    Write the current scan header to the `fd` file descriptor. If no argument or a NULL argument is given, the output goes to the output stream of the object.

39. **int sgetval(char *Key, float *Val)**

    Same as `ggetval` but operates on the current scan header.

40. **int sgetval(char *Key, double *Val)**

    Same as `ggetval` but operates on the current scan header.

41. **int sgetval(char *Key, int *Val)**

    Same as `ggetval` but operates on the current scan header.

42. **int sgetval(char *Key, char *Val)**

    Same as `ggetval` but operates on the current scan header.

43. **int sputval(char *Key, float v)**

    Same as `gputval` but operates on the current scan header.

44. **int sputval(char *k, double v)**

    Same as `ggetval` but operates on the current scan header.

45. **int sputval(char *k, int v)**

    Same as `ggetval` but operates on the current scan header.

46. **int sputval(char *k, char * v)**

    Same as `ggetval` but operates on the current scan header.

47. **int sputhistory(char *)**

    Same as `gputhistory` but operates on the current scan header.

48. **SCANINFO scaninfo**

    The `SCANINFO` object used by this object to navigate in the scans of the data base. The methods of this "internal" object are available (breaking the classical object oriented design!).

    This object is described elsewhere.

2.2 The `LTAVIEW` object

This section describes the `LTAVIEW` class derived from the `LTAFMT` object. This gives a higher level "view" of the database, with selections on scans/source applied. Optionally, this can also give a view of the data with a higher integration in time.

1. **LTAVIEW();**

   The default constructor for instantiating a blank object.

2. **LTAVIEW(char *FileName,char *OutFile=NULL, char *Obj=”.”);**

   Overloaded constructor. Takes the input file name, output filename and the name of the source. If the second and third argument is not given, the above mentioned defaults would be taken.

3. **void reset(char *FileName,char *OutFile=NULL, char *Obj=”.”);**

   Method to reset the internals of the object. The arguments has the same meaning as in the above constructor.

4. **newobj(char *Obj=”.”);**

   Method to just reset the object name without resetting all the internals. After a call to this, the the file pointer will sit at the first scan of the source selected.

5. **LTAVIEW();**

   The default destructor.

6. **int getnscans();**

   Returns the number of scans detected in the data base subject to the selection on source name given in one of the above methods.

7. **int skipscan(int n);**

   Method to jump to the n^{th} scan of the selected source.

8. **int rd_data(char *);**

   Same as the `LTAFMT::rd_data(char *)` method except that this will be sensitive to the source selection.

9. **int rd_data();**

   Same as the `LTAFMT::rd_data()` method except that this will be sensitive to the source selection.

10. **int avgb(char *, int *NRec, float Tint=0.0);**

    Same as `rd_data` method except that this will integrate for `Tint` seconds on the data. The number of records used to generated the integration is returned in `NRec`.

11. **int avgn(char *, struct fftmac *, int *,float Tint=0.0);**

    Same as `avgb` except that the visibilities would be normalized with the geometric mean self correlations of the two antenna which form a baseline.

12. **void getHMS(double t,float *hms);**

    Converts the time in seconds given in to hours,minutes and seconds.

13. **void initTime();**

    Initializes the object with the start time from the latest scan header. This time is used only to determine the data and year for calculation of LST later.

14. **double getLST(char *buf=NULL);**

    Return the LST for the data record pointed to by `buf`. If `buf` is not given or points to NULL, the internal data buffer would be used.

3 C Library for Handling Database

This section describes the C interface to the LTA database. C programmers should note that C applications can open only one LTA file in a single program.

3.1 The C Library

1. **void beginheader_(char *, char *)**

   Begin reading *GLOBAL* Header to initialize all Keywords and their corresponding values. The first argument is the name of the input file and the second argument is the name of the output file.

2. **void fbeginheader_()**

   A function callable from FORTRAN. This does the job of beginheader_() above.

3. **int skipscans_(int *n)**

   Skip `n` scans in a LTA file.

4. **int ggetival_(char *Key, int *Valal)**

   Get an integer value associated with the keyword `Key` from the *GLOBAL Header* into the variable `Val`.

5. **int ggetfval_(char *Key, float *Val)**

   Get a floating point value associated with the keyword `Key` from the *GLOBAL Header* into the variable `Val`.

6. **int ggetdval_(char *Key, double *Val)**

   Get a double precession value associated with the keyword `Key` from the *GLOBAL Header* into the variable `Val`.

7. **int ggetsval_(char *Key, char *Val)**

   Get a character value associated with the keyword `Key` from the *GLOBAL Header* into the variable `Val`.

8. **int gputival_(char *Key, int *Val)**

   Converts the `int` value `Val` to ASCII string and puts as the value of the keyword `Key` in the *GLOBAL Header*.

9. **int gputfval_(char *Key, float *Val)**

   Converts the `float` value `Val` to ASCII string and puts as the value of the keyword `Key` in the *GLOBAL Header*.

10. **int gputdval_(char *Key, double *Val)**

    Converts the `double` value `Val` to ASCII string and puts as the value of the keyword `Key` in the *GLOBAL Header*.

11. **int gputsval_(char *Key, char *Val)**

    Puts the null terminated string `Val` as the value of the keyword `Key` in the *GLOBAL Header*.

12. **int gputhistory_(char *str)**

    Write the history string `str` in the*GLOBAL Header* in the memory.

13. **int sgetival_(char *Key, int *Val)**

    Get an integer value associated with the keyword `Key` from the *SCAN Header* into the variable `Val`.

14. **int sgetfval_(char *Key, float *Val)**

    Get an float value associated with the keyword `Key` from the *SCAN Header* into the variable `Val`.

15. **int sgetdval_(char *Key, double *Val)**

    Get a double-precision value associated with the keyword `Key` from the *SCAN Header* into the variable `Val`.

16. **int sgetsval_(char *Key, char *Val)**

    Get a string value associated with the keyword `Key` from the *SCAN Header* into the variable `Val`.

17. **int sputival_(char *Key, int *Val)**

    Converts the `int` value `Val` to ASCII string and puts as the value of the keyword `Key` in the *SCAN Header*.

18. **int sputfval_(char *Key, float *Val)**

    Converts the `float` value `Val` to ASCII string and puts as the value of the keyword `Key` in the *SCAN Header*.

19. **int sputdval_(char *Key, double *Val)**

    Converts the `double` value `Val` to ASCII string and puts as the value of the keyword `Key` in the *SCAN Header*.

20. **int sputsval_(char *Key, char *Val)**

    Puts the null terminated string `Val` as the value of the keyword `Key` in the *SCAN Header*.

21. **int sputhistory_(char *str)**

    Write the history string `str` in the *SCAN Header* in the memory.

22. **int wt_data_(char *data)**

    Write a data record of length equal to the record length as given in the GLOBAL Header.

23. **int rd_data_(char *data)**

    Read a data record of length equal to the record length as given in the GLOBAL Header.

    It returns one of the following values:

    - NEWSCAN
      This is returned when the read operation read a *SCAN Header*.
    - DATAREC
      This is returned when the read operation read a data record.
    - EOF
      This is returned when the read operation hits the end-of-file.
    - UNKNOWNREC
      This is returned when the read operation read a record of unknown type.

24. **void gprintheader_(void)**

    Prints the *GLOBAL Header* as ASCII text on the standard output.

25. **void gdelheadrec_(int *n)**

    Delete the `n`th entry in the global header.

26. **void gwriteheader_()**

    Write the *GLOBAL Header* on to the output stream.

27. **void sprintheader_(void)**

    Write the *SCAN Header* on the standard output.

28. **void swriteheader_()**

    Write the *SCAN Header* on to the output stream.

29. **void sdelheadrec_(int *n)**

    Delete the $\mathtt{n}^{th}$ entry in the current *SCAN Header*.

30. **void ieee_compress_(void *cmplx, void *comp, int *ncmplx)**

    The data compression algorithm. The data in the data buffer of a LTA formatted file is compressed using this algorithm. *cmplx* is a pointer to a buffer having complex visibilities for all baselines for all channels. *ncmplx* is the total number of complex numbers in *cmplx*. The compressed buffer is returned in *comp*. *cmplx* and *comp* can point to the same buffer, in which case this call will do a in-place compression.

    Memory for *comp* must be allocated by the caller of this routine.

31. **void ieee_uncompress_(void *comp, void *cmplx, int *ncmplx)**

    The data de-compression algorithm. *comp* points to an uncompressed buffer with *ncmplx* complex numbers in it. The de-compressed buffer is returned in *cmplx*. Memory for *cmplx* must be allocated by the caller of this routine.

4 Appendix A: List Of Functions Available Under `glib.c` (General Library)

1. **void *getmem(int n, char *msg)**

   Does an `malloc(n)` and checks for errors. If an error occurs, `msg` is printed on the standard output.

2. **void getist_(char date[], int ist[])**

   Convert the `date` string to Hours, minutes and seconds in the array `ist`. The `date` string should be in the format return by the UNIX ``‘‘date’’`` command.

3. **void fLoadAntTab(char *FileName, struct AntCoord *Tab, int n)**

   Load the Antenna Table from the File `Filaname` into `Tab`. `n` is the number of antennas to be loaded.

4. **void LoadAntTab(struct AntCoord *Tab, int n)**

   Load the Antenna Table from the *GLOBAL Header* into `Tab`. `n` is the number of antennas to be loaded.

5. **void CopyAntTab(struct AntCoord *OldTab, struct AntCoord *NewTab, int n)**

   Copy `OldTab` to `NewTab` of size `n`.

6. **void ReplaceAntTab(struct AntCoord *Tab, int n)**

   Replace the Antenna Table in the *GLOBAL Header* by the new table `Tab`.

7. **void UpdateAntTab(char *FName, struct AntCoord *Tab, char *Tflag, int n)**

   Reads a new antenna table from a file `FName` and modifies the antenna table in the header according to the flag Tflag

   - Tflag=”New”
     Replace the old table by the new one in the header. On exit, `Tab` has the new table
   - Tflag=”Add”
     Add the new table to the old table in the header. On exit, `Tab=NewTab`
   - Tflag=”Delta”
     Replace the old table by the new table in the header. On exit, `Tab=NewTab - OldTab`.

   On exit, the caller can directly use the Tab to apply to the and the data will then reflect the operation defined by Tflag IMPORTANT: For Tflag=”New”, to truly reflect the action, the OldTab has to be ”un-applied” before NewTab can be applied

8. **GetUa(struct AntCoord, float U,float SH,float CH)**

   Return the `U` co-ordinate. `SH` has *sin(HA)* and `CH` has *cos(HA)*.

9. **GetVa(struct AntCoord COORD,float V,float SH,float CH,float SD,float CD)**

   Return the `v` co-ordinate. `SH` has *sin(HA)* and `CH` has *cos(HA)*. `SD` and `CD` has *sin(Dec)* and *cos(Dec)* respectively.

10. **GetWa(sturct AntCoord COORD,float W,float SH,float CH,float SD,float CD)**

    Return the `W` co-ordinate. `SH` has *sin(HA)* and `CH` has *cos(HA)*. `SD` and `CD` has *sin(Dec)* and *cos(Dec)* respectively.

11. **fGetUVWa(struct AntCoord COORD, struct Coord3D uvw, float SH, float CH, float SD, float CD)**

    Returns the `(U,V,W)` co-ordinates. `SH` has *sin(HA)* and `CH` has *cos(HA)*. `SD` and `CD` has *sin(Dec)* and *cos(Dec)* respectively.

    `GetUa, GetVa, GetWa, fGetUVWa` are all macros defined in `glib.h`.

12. **void warning(char *prog, char *msg)**

    Print a warning message `msg` qualified by the program name `prog` on the standard error.

13. **void linreg(float *x, float *y, int n, float *slope, float *intercept)**

    Linear regression for the least square fit for x,y.

14. **int month(char *m)** Returns an integer between 1-12 for the month no.

15. **int getchanlist_(int *c)**

    Get the list of available Channels into the array `c`.

16. **int getbaselist_(int *c)**

    Get the list of available baselines into the array `c`. The array should be at least as long as the total number of baselines in the database.

17. **int gettimestamp_(char *data)**

    Read the timestamp from the data record pointed to by `data`. `data` must point to the data of the first channel of the first baseline. This is generally achieved by passing `data+DATA_OFF` where `DATA_OFF` is read from the *GLOBAL Header*.

18. **int getnfft_()**

    Get the number of fft channels.

19. **int getfftmac(struct AntCoord *Tab, struct fftmac *corr)**

    Loads the mapping between FFT and MAC. It also loads the table of fixed delay, RF freq., side band, and polarization of the FFT pipe lines.

20. int **allocarray(struct fftmac *);**

    Allocate the internal buffer of the *fftmac* structure. This is generally for internal use and is used by *getfftmac*.

21. **int mkrecl_(int *nb, int *nc)**

    Returns the records the length given the number of baselines and channels.

22. **int getnchan(),getnbase();**

    Get the number of channels and baselines in the database.

23. **int getbaselist(int **List),getchanlist(int **List),getantlist(int **List);**

    Get the list of indices of baselines, channels or antennas. The argument `List` must be initialized to NULL. Upon return, the the `List` will point to an array of integers the length of which is the return value of these functions.

24. **void upcase(char *String);**

    Convert the given string to upper case.

25. **char *EatBlanks(char *String);**

    Deletes blanks from the given string and return the pointer to string.

26. **void averageupdate_(int *c0, int *c1, int *c2);**

    Updates the header of an LTA file to reflect averaging of channels. The header is set to reflect that all channels starting from `c0` up till `c1` with an increment of `c2` have been averaged.

27. **void average_(char *Data,int *c0, int *c1, int *c2, int *nb, int *nc, int *DataOff);**

    Perform the averaging of the channels. It performs an in-place averaging of channels in the data buffer `Data`, starting from `c0` to `c1` with an increment of `c2`. `nb` is the number of baselines and `nc` is the number of channels in the data (before averaging). `DataOff` is the offset in the data buffer from where the visibility data starts.

28. **void clipAverage_(char *,int *, int *, int *, int *, int *, int *);**

    Same as `average_` except that it will also attempt to clip "highly" discrepant points before averaging.

29. **int normalize(float *Data, int NChan, struct fftmac *fm);**

    Given the data buffer `Data`, the total number of channels in the database (`NChan`) and the sampler-MAC mapping via `fm`, this routine normalizes the visibility data with geometric mean of the self correlation amplitudes.

30. **void putmodparam(float *data, int nant, float *param);**

    Put the parameters (like delay, delay rate, etc.) in the data buffer parameter section. `data` is the data buffer, `nant` is the total number of logical antennas (samplers) in the database. `param` is an array of parameter values.

31. **int ReMap(int *dBList, int NdB, int *UserList, int Nu);**

    Remap the user supplied list (which is often 0-relative) to database indices.

32. **int newmklist(char *Str, int **List);**

    Given a string of the format I0:I1:I2 in `str`, it builds a list of integers starting from I0 up to I1 with an increment of I2. `List` must be initialized to NULL and the function will allocate memory to it using `malloc`.

33. **int setBit(int N, char **List, int *Size);**

    Set the N^{th} bit in the bit list `List`. The current size of the bit list is returned in `Size`. If the `List` is not long enough to hold the N^{th} bit, it is resized to a length ¿= N using `realloc`.

    In the first call to this function, `List` must be initialize to NULL.

34. **int toggleBit(int N, char **List, int *Size);**

    Toggle the N^{th} bit of the bit list `List`. The current size of the bit list is returned in `Size`. If the `List` is not long enough to hold the N^{th} bit, it is resized to a length ¿= N using `realloc`.

    In the first call to this function, `List` must be initialize to NULL.

35. **void fprtBits(FILE *fd,char *List, int Size);**

    Write the list of bit numbers of `List` which are set to 1 to the file descriptor `fd`. `Size` is the size of `List`.

36. **int bitCount(char *list, int Size);**

    Return the number of bits which are set to 1 in `List` of size `Size`.

37. **int MkBaselines(char *BName, LTAFMT &db, BitField &List)**

    Interprets and converts the string `BName` to the bit field object `List`. Bits in `List` corresponding to the selected baseline numbers are set to 1. `db` is a reference to the `LTAFMT` object.

    Before using the `BitField` object, it must be resized to zero by using List.resize(0).

    This function is callable from `C++` only (since it uses `C++` object `LTAFMT`).

38. **int MkAntNo(fftmac &fm, char *AName, BitField &AList)**

    Interprets and converts the string `AName` to a bit field object `AList`. Bits in `Alist` corresponding to the sampler numbers of the selected antennas are set to 1. `fm` is a reference to `struct fftmac`.

    Before using the `BitField` object, it must be resized to zero by using AList.resize(0).

    This function is callable from `C++` only.

39. **void toIntList(BitField &BitList, int **IntList);**

    Convert the bit list from `BitList`, into integer array and return the array in `IntList`. `IntList` must be initialize to NULL before calling this function. Upon return, `IntList` contains the list of bits which were set to 1 in the bit field `BitList` and the size of `IntList` is the return value of this function.

    Memory for `IntList` is allocated using `malloc`.

40. **void MkFullAntName(struct fftmac *fm,int Ant, char *Name);**

    Given the pointer to the `fftmac` structure, it returns the fully qualified name of antenna `Ant` in `Name`. A fully qualified antenna names is of the form AAA-BBB-PPP where AAA is the antenna names, BBB is the side band names and PPP is the name of the polarization channel. The string `Name` should be long enough to hold the fully qualifed name (which, present would be of size 11 bytes + 1 byte for the terminating NULL character).

41. **void ReArrangeBase(float *Data, struct fftmac *fm, int NChan);**

    Rearrange the baseline information in structure `fm` such that the index of the first antenna in the all baselines is smaller than the index of the second antenna. The phase of the visibility in such a re-arrangement is consistently modified.

42. **int genfree(char *Msg, int n,...);**

    Generic function to free memory from a list of pointers. The memory is freed using the `free`. `Msg` is the message to be printed on the standard error stream and the list of pointers is supplied from the second argument onwards. The last argument must be NULL.

43. **FILE *openfd(char *Name, char *Permissions);**

    Open a file by the name `Name` with the permissions given in the string `Permissions` (with the same meaning as in the system call `fopen`). If the `Name` begins with the character '—', the file is opened as a pipe. The file pointer is the return value. If the return value is NULL, the function failed to open the file/pipe.

44. **int closefd(FILE *);**

    Close the file pointer opened using `openfd`.

45. **int IsNANorINF(float);**

    Returns 1 if the given floating point number is a NAN (Not A Number) or Inf (Infinity), and 0 otherwise.