The GMRT LTA format

Jayaram N Chengalur

26/Feb/2002

1 Introduction

Interferometric data at the GMRT is stored in a format called the `lta`[1] format. This document is a detailed specification of the lta format and is targeted at people who would like to write/modify offline data analysis software.

As shown in Figure 1, the lta format is record oriented, i.e. it consists of an integer number of fixed length records[2]. The file starts with a global header, followed by a scan header. Data for the first scan begins immediately after the end of the scan header. The second scan header begins immediately after the last data record of the first scan, and so on. The sizes of the global header and the scan headers are guaranteed to be integer multiples of the record length. All the visibility (and associated) data corresponding to a single timestamp are placed in a single record, i.e. each data record corresponds to a unique timestamp. In this sense the lta file is in time-baseline order.

The first 4 bytes of almost[3] every record is a signature. These 4 bytes specify the type of the record. At the moment there are 3 valid signatures, viz.

1. HDR Indicates that the record is the first record of a global header.
2. SCAN Indicates that the record is the first record of a scan header.
3. DATA Indicates that the record is a data record.

For SCAN and DATA type records the first few bytes immediately after the signature indicate the scan number and the record number in that scan. See figure 1 and the following sections for more details.

[1]This name is derived from that of the last stage of the correlator, viz. the Long Term Accumulator.

[2]This is not strictly true. Programmers should be aware that in case the DAS chain crashes one could be left with an lta file whose last record is incomplete.

[3]The only exceptions are the second and succesive records of global and scan headers. These records do not begin with a signature.

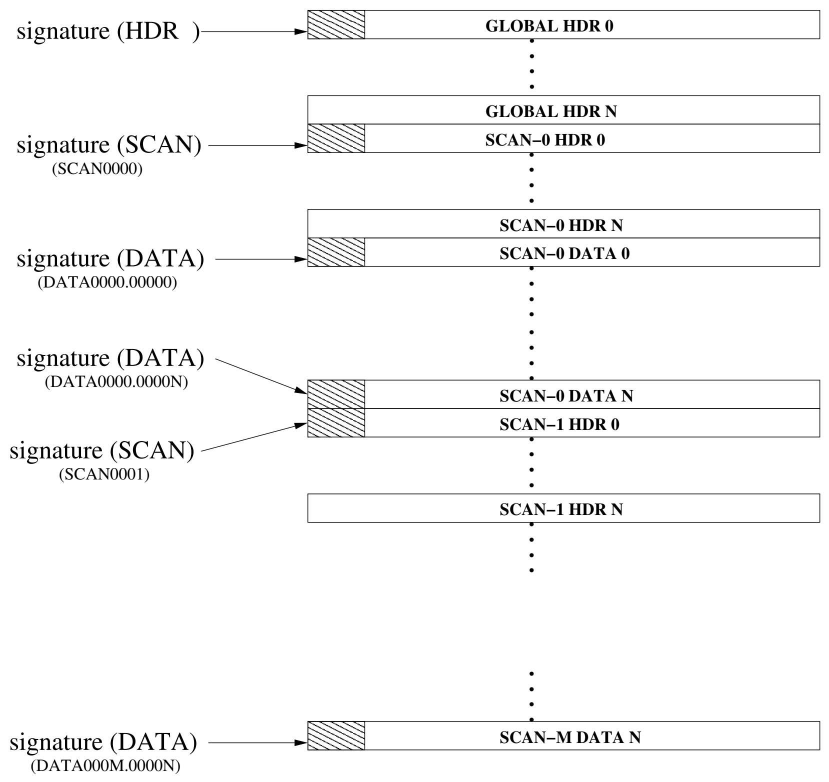

Figure 1: Overview of the LTA file structure. LTA files are record based, i.e. they consist of an integer number of fixed length records. Each file starts with a global header followed by a scan header. Data for a scan starts immediately after the end of the scan header. After the last data record of the first scan one has the scan header for the next scan, and so on. The first record of the global header begins with the signature "HDR ". The first record of each scan header begins with the signature "SCAN". The next 4 bytes indicate the scan number starting with 0000. Each data record begins with the signature "DATA". The next 11 bytes are MMMM.NNNNN where MMMM is the scan number and NNNNN (starting with 00000) is the running number the given data record in this scan.

2 The Global Header

The global header contains information on both the organization of the lta file as well as global details of the observation, i.e. things like the number of baselines and channels, which do not change from one scan to another. This header consists of two parts, an "ASCII header" and a "binary header". In the current version, the ASCII header is complete in itself, i.e. offline programs can (and in fact do) ignore the binary header. However, this may be depreciated in the future. The binary header is not complete in itself, i.e. if one reads only the binary header one will not be able to parse the lta file. In particular the binary header only contains details regarding the observation parameters and has no information on the layout of the lta file. In general the binary header has more detailed information on the observation set up than is available in the ASCII header.

The ASCII header itself consists of a number of 80 byte blocks. The first 80 bytes of the ASCII header are

HDR   REC_LEN HDR_RECS   AHDR_RECS

where REC_LEN is the record length in bytes (i.e. all records in this file are of length REC_LEN bytes), HDR_RECS is the total number of records that the global header occupies, and AHDR_RECS is the total number of records that the ASCII header occupies. The remaining HDR_RECS-AHDR_RECS records consist of the binary header. For example, a global header starting with

HDR 1013032   2   1

indicates that the record length in this file is 1013032 bytes, and that the global header is 2 records long, of which 1 record (the first record) is the ASCII header and 1 record (the second record) is the binary header. Programs which parse this should only look for three white space separated integers after the string "HDR", and not for values starting at a fixed byte offset.

In the ASCII header, 80 byte blocks that start with the character "*" are comments. Comments are generally used to indicate the start and end of different sections of the ASCII header. For example the comment "*{ Init.def" indicates the start of the "Init" section of the ASCII header, while the comment "*} Init" indicates the end of the "Init" section of the ASCII header. Apart from these comments, the ASCII header consists of (keyword, value) pairs. The format is

KEYWORD = VALUE

Keywords are 8 characters or less in length. The 9th byte is always "="and the value always starts from the 11th byte. The value could contain white spaces, programs should in general read all bytes up to the last non white space byte.

The various keywords and their meanings are given in section 5. The end of the ASCII header is marked by an 80 byte block starting with the string "END_OF_HEADER". The rest of the ASCII header record is filled with white space (although programs should not depend on this.). Note that since it is not necessary for the record length to be a multiple of 80 bytes, if the ASCII header occupies more than one record one could have an 80 byte block which is split across two adjacent records.

The binary header contains the "CorrType" structure that is used by the DAS programs. See the file "newcorr.h" (Appendix 7) for details. Basically if one reads sizeof(CorrType) bytes from the start of the binary header into an object that has been declared to be of type CorrType then one has recovered the CorrType structure that was used by the DAS programs which generated the given lta file. Note that if the endian type of the machine on which one is working is different from the one on which the lta file was recorded one will have to appropriately flip the bytes of each field in the CorrType structure. (The endian type of the lta file is recorded in the ASCII header).

3 The SCAN header

The scan header contains information that could change from scan to scan, i.e. things like the source name (and other source related parameters), the observing frequency, etc. Like the global header, the scan header also consists of two parts, the ASCII scan header and the binary scan header. The format of the ASCII scan header is identical to that of the ASCII global header, i.e. it too consists of 80 byte blocks of (keyword,value) pairs. Also, as in the ASCII global header, a "*" at the start of an 80 byte block denotes a comment. Comments in the ASCII scan header are generally of of the form"*{" and "*}", i.e. they are used to demarcate different sections.

The first 80 bytes of the ASCII scan header are

SCANMMMM   HDR_RECS AHDR_RECS

where MMMM is a integer indicating the scan number in the file. The first scan would have number 0000, the second 0001 and so on. HDR_RECS is the total number of records in the scan header and AHDR_RECS is the total number of records in the ASCII scan header. The end of the

ASCII scan header is marked by an 80 byte block starting with the string "END_OF_HEADER". The rest of the scan header record is filled with white space (although programs should not depend on this.). Note that, as discussed for the ASCII global header, if the ASCII scan header occupies more than one record, one could have an 80 byte block which is split across two adjacent records. The keywords that occur in the scan header are listed in section 6.

The binary scan header contains the "ScanInfoType" structure used by the DAS programs. See the file "newcorr.h" (Appendix 7) for details. Basically if one reads sizeof(ScanInfoType) bytes from the start of the binary header into an object that has been declared to be of type ScanInfoType then one has recovered the ScanInfoType structure that was used by the DAS programs which generated the given lta file. Note that if the endian type of the machine on which one is working is different from the one on which the lta file was recorded one will have to appropriately flip the bytes of each field in the ScanInfoType structure. (The endian type of the lta file is recorded in the global ASCII header).

4 Data Records

All the data acquired at a given time stamp are placed in a single record. The first 4 bytes of any data record is the string "DATA". The next 11 bytes are a string of the sort "MMMM.NNNNN", where MMMM is the scan number of this data record and NNNNN is the record number in this scan. For example, the first data record in the first scan would have the number "0000.00000", the second data record would have the number "0000.00001" and so on.

Each data record contains the timestamp[4], the record weight[5], flags, model parameters and the visibility data itself. The weight and time stamp are in double (8 bytes) format. The model parameters are the DataParType structure used by the DAS programs (see "newcorr.h" (Appendix 7) for details). The flags are described in section 5. The visibility data is in the form of two doubles (real and imaginary) per baseline per channel.

Information as to where in each data record these parameters can be found is given in the ASCII header. For example, the timestamp can be found at

[4]This is the IST time at the middle of the first STA cycle of this data record.

[5]This is the number of 0.128 sec LTA cycles that have been integrated to form this data record.

the byte offset TIME_OFF (from the start of each data record) and consists of TIMESIZE bytes. (TIME_OFF and TIMESIZE are keywords in the global ASCII header. All such keywords are listed in Appedix 5). Similarly the data starts at byte offset DATA_OFF and consists of DATASIZE bytes. As shown in Figure 2 in each record the data is arranged in baseline-channel order. That is at offset DATA_OFF one will have the visibility for channel 0 of baseline 0. The next datum will be channel 1 of baseline 0, and so on until one reaches the last channel of baseline 0. Then one has channel 0 of baseline 1 and so on until one reaches the last channel of the last baseline. The composition of the different baselines is specified in the global ASCII header (i.e. one has keywords of the sort BAS000 to BASNNN which specify the antennas (strictly speaking the sidebands and polarizations) which baselines 0 to NNN are composed off.)

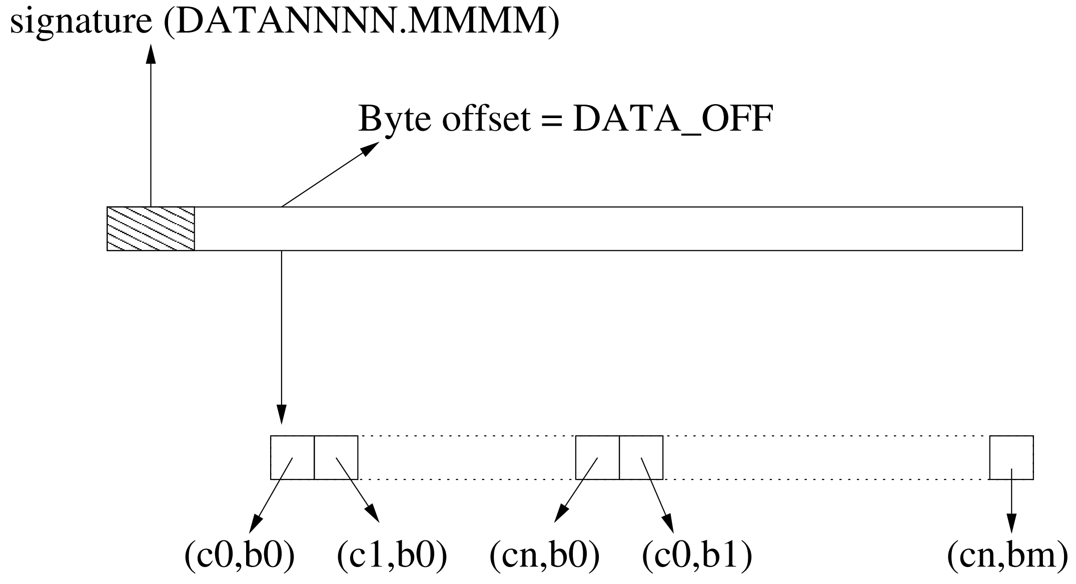

Figure 2: The structure of data records. The first 4 bytes are always “DATA”. The next 11 bytes are of the form MMMM.NNNNN where MMMM is the scan number (starting with 0000) and NNNNN is the number of the data record in this scan (starting at 00000). The visibility data starts at byte offset DATA_OFF in the record, and is in baseline-channel order. See the text for more details.

5 Appendix ASCII Global Header

```
**{ Init.def                                            ! start of Init section
RECL         =  1013032                                 ! record length (bytes)
HDR_RECS     =  2                                       ! number of header records
REC_FORM     =  PROJ_DATA                               ! type of recorded data (CHECK!)
OBS_MODE     =  pre-rel                                 ! observing mode
VERSION      =  2.0pre : MONLOG                         ! DAS and LTA version
D_TYPE       =  char short int long float double        ! data types used in file
D_SIZE       =  1 2 4 4 4 8                             ! sizes of these data types (bytes)
D_ALIGN      =  1 2 4 4 4 8                             ! alignment of this data types
INT_REP      =  Twos Comp                               ! data representation used for integers
FL_REP       =  IEEE                                    ! data representation used for floats
BYTE_SEQ     =  Big Endian                              ! byte sequence
*} Init                                                 ! end of Init section

*{ AddCorr.def                                          ! start of AddCorr section
ANTENNAS     =  30                                      ! total number of antennas in file
SAMPLERS     =  60                                      ! total number of samplers in file
BASELINE     =  930                                     ! total number of baselines in file
CHANNELS     =  128                                     ! total number of channels in file
FLG_OFF      =  80                                      ! offset at which flag tables starts
FLG_SIZE     =  59656                                   ! total size of flag tables (bytes)
FLGRECOF     =  80                                      ! offset at which per record flag starts
FLGRECSZ     =  4                                       ! size of per record flag
FLGANTOF     =  84                                      ! offset at which per antenna flags start
FLGANTSZ     =  4                                       ! size of the per antenna flag table (bytes
                                                        ! (one bit per antenna)
FLGSMPOF     =  88                                      ! offset at which per sampler flags start
FLGSMPSZ     =  8                                       ! size of the per sampler flag table (byte)
                                                        ! (one bit per sampler)
FLGBASOF     =  96                                      ! offset at which per basline flags start
FLGBASSZ     =  120                                     ! size of the per baseline flag table
                                                        ! (one bit per baseline)
FLGDATOF     =  216                    ! offset at which the per datum flags start
FLGDATSZ     =  59520                  ! size of the per datum flag table
                                       ! (4 bits per datum)
TIME_OFF     =  59736                  ! offset at which the time stamp starts
TIMESIZE     =  8                      ! size of the time stamp (bytes) (double)
WT_OFF       =  59744                  ! offset at which the record weigth starts
WT_SIZE      =  8                      ! size of the weight (bytes) (double)
PAR_OFF      =  59752                  ! offset at which the model parameters start
PAR_SIZE     =  960                    ! size of the model parameter structure
DATA_OFF     =  60712                  ! offset at which the visibility data starts
DATASIZE     =  952320                 ! size of the visibility data (bytes)
DATAFMT      =  COMPL.64               ! data format (8 byte complex)
*} AddCorr                             ! end of AddCorr section

*{ Corrsys.def                         ! start of Corrsys section
VERSION      =  CORR30                 ! DAS version
CLOCK        =  32000000.000000 Hz     ! clock speed (Hz)
STA          =  8192 Cycles            ! FFT cycles in 1 STA (CHECK!!)
STATIME      =  132096 usec            ! Length of 1 STA cycle (micro seconds)
T_UNIT       =  1.000000 sec           ! Time unit (unused?)
ANTS         =  30                     ! Max antennas in the correlator
F_ENABLE     =  60                     ! Number of samplers in the correlator
X_ENABLE     =  465                    ! Number of MAC chips in the correlator
CHAN_MAX     =  128                    ! Max chans in the correlator
POLS         =  2                      ! Max pols in the correlator
*} Corrsys                             ! end of Corrsys section
```

```
**{ Antenna.def         ! start of Antenna section
! This section gives the antenna co-ordinates. The units are meters.
! The co-ordinate system is the usual right handed one used in radio
! astronomy, see eg. Thompson, Moran & Swenson. The last two columns
! give the instumental delays, also in meters. There is one delay per
! polarization.
*AntId   =   Ant      bx          by          bz        delay0       delay1
ANT00    =   C00     6.95       687.88      -20.04     -497.89      -497.89
.
.
.
ANT29    =   W06   -3102.11   -11245.60   8916.26   -29266.87   -29266.87
**} Antenna.def         ! end of Antenna section

**{ Bandnames.def                       ! start of Bandnames section
BAND00              =   USB-130
BAND01              =   USB-175
BAND02              =   LSB-130
BAND03              =   LSB-175
*} Bandnames                            ! end of Bandnames section

**{ Sampler.def         ! start of the Sampler section
! This section gives the sampler conectivity. That is it
! specifies which sideband of which antenna is connected to
! a given sampler and FFT pipeline.
*SampId   =   Ant   BandId    FftId
SMP000    =   C12   USB-130   000
SMP001    =   C12   USB-175   001
.
.
.
SMP059    =   C13   USB-175   059
**} Sampler.def         ! end of the Sampler section
```

```
**{ Baseline.def        ! start of Baseline section
! This section gives the baseline composition. That is it tells you
! which sidebands of which antennas comprise a given baseline. Both
! the numbers and names of the sidebands and antennas are given. The
! order of baselines here is the same as the order in which data
! is sorted in each data record. That is the data for baseline 0
! (BAS000 in this table) appears at the start of the data block
! data for BAS001 appears at the end of the data for BAS000 and so on
*BaseId  =  A0   B0   A1   B1   SMP0   SMP1   Ant0   Band0     Ant1   Band1
BAS000   =  04   00   29   00   032    008    C04    USB-130   W06    USB-130
BAS001   =  04   01   29   01   033    009    C04    USB-175   W06    USB-175
.
.
.
BAS929   =  12   01   12   01   059    059    C13    USB-175   C13    USB-175
**} Baseline.def        ! end of Baseline section

**{ Corrsel.def                    ! start of the Corrsel section
MODE           =   0               ! DAS mode
LTA            =   128             ! STA cycles averaged in one record
SAMP_NUM       =   0:59:1          ! samplers whose data is in the file
CHAN_NUM       =   0:127:1         ! channel numbers whose data is in the file
*} Corrsel                         ! end of the Corrsel section
```

6 Appendix ASCII Scan Header

```
HDR_RECS       =  2                          ! numer of records in scan header
*{ SubArray0.def                             ! start of Subarray section
ANTMASK        =  3fffffff                   ! antennas in subarray
BANDMASK       =  0003                       ! sidband(s) in subarray
SEQ            =  0                          ! Project Seq No. (Useful to DAS only)
PROJECT        =  SYSTEM TEST                ! Project title
CODE           =  SYS                        ! project Code
OBJECT         =  3C48                       ! source Name
RA-DATE        =  24.452596                  ! source RA (mean)
DEC-DATE       =  33.170747                  ! source DEC (mean)
MJD_SRC        =  0.000000                   ! MJD of source co-ordinates
DRA/DT         =  0.000000                   ! rate of change of source RA
                                             ! (for solar system objects)
DDEC/DT        =  0.000000                   ! rate of change of source DEC
                                             ! (for solar system objects)
F_STEP         =  7812.500000                ! channel width (Hz)
RF             =  325000000 325000000        ! RF of channel 0 (Hz)
FIRST_LO       =  255000000 255000000        ! First LO (Hz)
BB_LO          =  70000000 70000000          ! Baseband LO (Hz)
NET_SIGN       =  1 1 -1 -1                  ! channel spacing=NET_SIGN*F_STEP
                                             ! per sideband per polarization
*} SubArray0                                 ! end of Subarray section

*{ ExtraScan.def                             ! begin of ExtraScan section
INTEG          =  16.908287                  ! integration time (sec)
DATE-OBS       =  Thu Feb 21 00:01:07 2002   ! Reference date and time
                                             ! NOTE: reference time always 00:00:00
                                             ! value here should be ignored
MJD_REF        =  52325.770833               ! The reference MJD
BAD_RECS       =                             ! list of bad records
BAD_ANTS       =                             ! list of bad antennas
BAD_SAMP       =                             ! list of bad samplers
BAD_BASE       =                             ! list of bad baselines
BAD_CHAN       =                             ! list of bad channels
*} ExtraScan                                 ! end of ExtraScan section
```

7 Appendix: The binary header ("newcorr.h")

```
enum  USB_130, USB_175, LSB_130, LSB_175, MAX_BANDS ;
   enum  RRLL, RRRL, RR__, MACMODES  ;
   enum  arar_arar, alal_alal, brbr_brbr, blbl_blbl,
   aral_brbl, aral_alar, brbl_blbr, arbr_albl, DpcMuxVals ;
   enum  NAMELEN=32, DATELEN=32 ;
   enum  MAX_ANTS=30, MAX_SAMPS=60, MAX_FFTS=MAX_SAMPS, MAC_CARDS=33,
   MAX_BASE=32*MAC_CARDS, MAX_CHANS=128, POLS=2
   ;
   /* be sure that MAX_FFTS and MAX_SAMPS are both even numbers !
*/
   enum  MAX_PROJECTS=500, MAX_SCANS=100  ;
   enum  LittleEndian=1, BigEndian=0 ;
   enum  TransitObs = 32768  ;
   enum  TimeSize=sizeof(double), WtSize = 2*sizeof(float),
   ActiveScans=8, DataFlagSize=ActiveScans*sizeof(int) ;
   typedef struct
   char name[4];
   unsigned char samp_id[MAX_BANDS] ;
   double bx, by, bz; /* metres */
   double d0[MAX_BANDS], p0[MAX_BANDS];
   /*
   samp_id is the local reference for sampler;
   corr->sampler[samp_id].dpc gives the absolute
   sampler channel number
   scan_id is dynamic and is only locally revelant
   to a specific program, pointing to an index
   in a scan table which the program may be maintaining.
   Others have global significance for all programs
   */
   AntennaParType;
   typedef struct
   int macs, channels, pols, sta, statime ;
   float f_step, clock; /* Hz */
   unsigned char dpcmux,clksel,fftmode,macmode ; /* replaces old
dummy int */
   CorrParType ;
   typedef struct
   int antmask, samplers, baselines, channels, lta;
   short bandmask, mode ;
   short chan_num[MAX_CHANS] ;
   double mjd_ref, t_unit ; /* (mjd_ref*t_unit), (timestamp*t_unit)
in sec */
   DasParType ;
   typedef struct
   char object[NAMELEN];
   struct  float i,q,u,v ;  flux ;
   double mjd0 /* fractional mjd, to which ra, dec refer */ ;
   /*
   mjd0 refers to the epoch of ra_app,dec_app.
   Note that the timestamp in data is wrt to the global
   reference time contained in daspar->mjd_ref
   */
   double ra_app,dec_app,ra_mean,dec_mean,dra,ddec ; /* rad, rad/s
*/
   double freq[2], first_lo[2],bb_lo[2]; /* Hz */
   double rest_freq[2], lsrvel[2] ; /* Hz, km/s */
   double ch_width ; /* Hz */
   int id, net_sign[MAX_BANDS], mode , dum1;
   unsigned int antmask; /* antennas to fringe stop */
   unsigned short bandmask, dum2;
   short calcode, qual ;
   SourceParType ;
   typedef struct
   char code[8], observer[NAMELEN], title[NAMELEN] ;
   unsigned int antmask ; /* antennas to record */
   unsigned short bandmask,seq;
   ProjectType ;
   typedef struct
   int status ;
   float t ; /* program dependent meaning !  */
   ProjectType proj ;
   SourceParType source ;
   ScanInfoType ;
   typedef struct  unsigned char ant_id, band, fft_id, dpc;  SamplerType;
```

```
  typedef struct
  SamplerType samp[2] ;
  unsigned char card,chip,pol,word_incr ;
  /* e.g., RRLL pol=1?  word_incr=2 represents RR component ;
  RRLL pol=0?  word_incr=2 represents LL component ;
  RRRL pol=0?  word_incr=2 represents RL component ;
  RR__ pol=0?  , word_incr=1
  */
  BaseParType ; /* try this structure instead of BaseType */
  typedef struct
  unsigned char endian,dummy[7];
  char version [NAMELEN ]; /* should begin with string "CORR" */
  char bandname[MAX_BANDS ][8];
  AntennaParType antenna [MAX_ANTS ]; /* Ant pos, freq & other config
*/
  SamplerType sampler [MAX_SAMPS ]; /* ant, band vs.  ffts */
  BaseParType baseline[MAX_BASE ]; /* Pair of ant, band */
  CorrParType corrpar; /* Max.  enabled mac_params */
  DasParType daspar; /* Actually useful params */
  CorrType;
  typedef struct  unsigned int ext_model,idle,stop,userflag ;  AntennaFlagType
;
  typedef struct  float phase, dp, delay, dd /* sec/sec */ ;  ModelParType
;
  typedef struct  float phase, dp, delay, dslope;  DataParType;
  /* units:  phase in radian, dp in radian/sec,
  delay in sec, dslope in radians/channel
  */
  typedef struct
  double t0 ; /* seconds wrt corr->daspar.mjd_ref */
  int ant_id, band ;
  ModelParType par ;
  ExtModelType ;
  # define AntennaTypeSize sizeof(AntennaType)
  # define MacFftTypeSize sizeof(MacFftType)
  # define SamplerTypeSize sizeof(SamplerType)
  # define DataParTypeSize sizeof(DataParType)
  # define CorrTypeSize sizeof(CorrType)
# define CorrSize sizeof(CorrType)
typedef struct
int in0,in1,out0,out1 ; IndexType ;
typedef struct
double bxcd,bycd,bzsd,fixed,phase, bb_lo,freq, f_step ;
GeomType ;
# define DAS_H_KEY 1030
# define DAS_D_KEY 1031
# define DAS_H0_KEY 1032
# define DAS_D0_KEY 1033
# define DAS_BUFSIZE 4096000
# define DAS_HDRSIZE 200000
typedef struct
int s0,s1, card; /* card points to relevant delay/FFT card */
/*
The two relevant samplers are given by
daspar.samp_num[s0] and daspar.samp_num[s1]
*/
int delay, p0, pstep0,p1,pstep1, fstc ; /* is int ok?  */
/* Do not plant delay difference between two streams here;
the difference must be handled as static difference
during initialisation
*/
float p2fstc, fstc_step ; /* is float fine :  sirothia 13oct */
FftDelayParType ;
typedef struct
double clock, t_update ;
double pc_time ;
double t_off; /* JNC 16Dec98*/
double delay_step, fstc_scale, nco_tick_time ;
int cycle, seq, cards ;
/*cycle = STA cycles between model update */
unsigned char dpcmux,clksel,fftmode,macmode ;
ModelParType par[MAX_SAMPS] ;
FftDelayParType fdpar[MAX_ANTS] ;
ModelInfoType ;
typedef struct
int active, status, scan, scan_off;
```

```
   CorrType corr ;
   ModelInfoType model ;
   char buf[DAS_HDRSIZE];
   DasHdrType ;
   enum  BufMarked=1, BufReady=1<<1, Rack0Bad=1<<2,Rack1Bad=1<<3,Rack2Bad=1<<4,
   Rack3Bad=1<<5,Rack4Bad=1<<6,Rack5Bad=1<<7, BufFinish=1<<8,
   MaxDataBuf=100
   ;
   enum  MAX_EVENTS=50000  ;
   typedef struct
   float t ;
   unsigned char type, cmd ;
   unsigned short seq ;
   int flag_num, scan_num ; /* indexes on AntennaFlag and ScanInfo
*/
   EventLogType ;
   typedef struct
   int t_off, wt_off, par_off, data_off, data_words ;
   short par_words, wordsize ;
   RecOffsetType ;
   typedef struct
   int off ;
   BaseParType base ;
   char name[12] ;
   MacWordType ;
   typedef struct
   unsigned char seq, scan ;
   int status,recl, seqnum ;
   RecOffsetType off ;
   MacWordType *mac ;
   char *buf ;
   RecBufType ;
   typedef struct
   int active,status;
   unsigned short events,flags,starts,stops ;
   CorrType corr ;
   AntennaFlagType flag[MAX_EVENTS][MAX_BANDS] ;
   EventLogType event[MAX_EVENTS] ;
```

```
ScanInfoType scaninfo[MAX_SCANS] ;
RecOffsetType offset ;
DataInfoType ;
typedef struct
int flag,rec,seqnum ;
unsigned short flag_seq, newstate ;
DataTabType;
typedef struct
int flag, blocksize, maxblocks, cur_block, first_block, cur_rec;
DataTabType dtab[MaxDataBuf];
char buf[DAS_BUFSIZE] ;
DataBufType ;
```