OBIT

Merx mollis mortibus nuper

Version: 1.2 July 31, 2010

W. D. Cotton

Contents

1 Introduction

OBIT is a software library package intended for astronomical, especially radio–astronomical applications. This document describes the high level view of the software system and is the fundamental definition of the tables used in the Obit software package.

Obit is an object–oriented set of class and utility libraries allowing access to multiple disk–resident data formats. In particular, access to either AIPS disk data or FITS files. Documentation of the Obit classes uses the doxygen documentation system and can be accessed starting at doc/doxygen/html/index.html.

2 Design Objectives

The principle motivation for Obit is to provide a software environment where I can develop algorithms and develop targeted production software for particular projects, especially wide area sky surveys. AIPS has gotten too unwieldy for new development. However, there are a number of technical problems for which this can be considered exploratory.

2.1 Multiple Persistent "Native" Data Forms

With proper layering and inheritence strategies, multiple external data forms can be used as "Native" data, i.e. do not need to be translated to/from another persistent data system. This requires that there be a common data (including calibration) model and that the data models can all be mapped onto that of the software (and vice versa). For the interferometric and image applications in Obit, the current implementation supports both AIPS and FITS files as internal formats. At the applications level, the only time the distinction is made is when the object is associated with a particular external AIPS entry or FITS file. Derived IO classes translate the disk resident structures to Obit's memory resident forms and back again.

2.2 Flexible Class System in c

Medium to large scale software packages are much more tractable using object-oriented methodology. In practice this means a class system supporting inheritance and polymorphism. The obvious candidates for this are C++ and Java. In practice, C++ software has serious robustness problems and it appears beyond my ability to write (actually to debug) C++ code to a sufficient standard. Java is a much better designed language but has a performance penalty of a factor of several.

The fallback position is a "roll my own" class system in ANSCI c. This seems to be a reasonable compromize between robustness and performance. However, the c family has issues, especially with memory usage which have been only partially solved in Obit.

Each Obit class has a global structure which contains class information, these include function pointers and a pointer to the parent class structure.

In practice, the class and inheritence system is implemented using include files and the c preprocessor. Each class definition header recursively includes its parents object definition, making all data members effectively "public". Member functions of the class can be either those of the parent class, set to those specific to the derived class, set to NULL, or new class function added. Private member functions are allowed by declaring them "static" inside the class implementation file. Class member functions can be invoked either explicitly or by referencing the function from the class structure.

This implementation has the drawback that a object must be explicitly included as arguments in calls to its function, this makes the relationship less obvious. The convention has been adopted to make the "self" object the first argument in the call sequence.

2.3 Robust Object Managment

The c language family has no garbage collection meaning that applications must do their own memory managment. A further complication is that the c family memory managment tools are very unforgiving and errors are frequently very difficult to locate. E.g. a second "free" of a block of memory causes no immediate problem but at some later, and usually unrelated point, the program will abort. Part of the solution is to adopt the glib memory managment routines - at least g_free does not abort given a null pointer.

Many objects contain substantial resources, memory, open files etc. which would be expensive and risky to frequently copy. The usual solution to this problem is to allow multiple pointers to the same object; thus an object can serve as a member of another object(s) as well as stand on its own. This aggravates a further problem that each object must be destroyed and its resources released EXACTLY once. The standard solution to this problem is "reference counting", each object keeps a count of the current number of references.

This solution has yet further consequences, a great deal of discipline is required to keep the reference count correct. The Obit solution to this problem is a pair of member functions for each class to reference and unreference an object. The reference function increments the objects reference count and returns a pointer to the object. The unreference function decrements the reference count, destroying the object if it hits zero and returns a NULL pointer. There is no explicit object destructor.

A further complication is that objects may be passed as a parent class. To allow type checking, each class has an "IsA" function to determine if an object is of the correct type and is valid. This is usually wrapped into a glib assertion that can be turned off in production (debugged) software.

2.4 Parallelization

The use of Fortran named commons in AIPS makes high level parallelization difficult in that environment. The use of multiple objects with independent resources reduces the difficulty. In addition, most classes have an ObitThread member to provide multi–threaded operation, locking and synchronism mechanisms. cfitsio, which is used for I/O to FITS files, is not thread–safe which limits parallel, multi–threaded usage to in–memory applications.

Multi-threading is implemented in the most CPU intensive operation by splitting the work on a single I/O buffer amoung various threads. This means fairly small work packets so the threading is done using gthreads thread pools. This is accessed via ObitThreadIterator.

2.5 Portability

The glib library was intended to reduce portability problems and many of its features are adopted.

2.6 Name Space Control

The name space is controlled by including the class name at the beginning of all public function names. All begin with "Obit".

2.7 Passing Control Parameters

One of the great causes of instability in software is the changing of call sequences through which control parameters are passed when these are changed. This problem is dealt with in Obit by means of the ObitInfoList which is basically an associative array (labeled, arbitrary pieces of information). Objects for which control parameters may be relevant have an ObitInfoList member. High level routines put control information into this container rather than passing it through call sequences. Lower level routines can then take the appropriate response to a missing entry, either assume a default value or treat the absence as an error condition.

2.8 Error handling

Trapping error conditions and giving a traceback to their cause is combined with informative and warning messages in the ObitErr class. An ObitErr has an error indication word and a message stack; each message has an associated error level. Utility macroes are provided such that when a routine detects a condition that it cannot handle, it set an error flag, registers an error message giving the software locations and returns. The calling routine can, deal with the problem and remove the error condition and messages, add additional diagnostic information and return, or note its location in the error log and return. Routines with an ObitErr argument should check that no prior error condition exists and return if one does. At the appropriate level, the contents of the message log can be displayed and any error condition cleared. At present, the output is to stdout or a specified log file, but more formal logging, such as AIPS logging, is easily implemented. The error object also contains a "prtLv" (print level) member that can be used to control the amount of diagnostic messages. The prtLv and logging to a file are enables using the ObitErrInit function.

3 Obit Software System

The Obit system maintains its basic information in global structures. These are initialized by a call to ObitSystemStartup and released by a call to ObitSystemShutdown.

3.1 General Software Issues

Obit is written in ANSI c of the glib dialect to assist in portability. Thus most data types are declared with the "g" prefix (e.g. gfloat = float). Also, g_malloc and g_free are used to allocate and free memory.

Each class and structure in Obit has an "IsA" function to test that the pointer passed is to a valid instance of that structure or class.

The glib library provides assertion macroes which are used to test for programming errors but the tests can be turned off in a production compilation. The assertion g_assert should be used with the class IsA test functions to assure that the inputs to each routine are valid. If the assertion fails, the program aborts and dumps core.

3.1.1 Obit Initialization/Shutdown

Persistent data forms are kept in a number of directories referenced by "disk" number. There are separate lists for AIPS and FITS data and these are initialized when Obit is initialized using the ObitSystemStartup function. If the relevant directories are given in standard Unix environment variables, they can be picked up automatically. This is illustrated in the following code fragment:

```
ObitSystem *mySystem;
ObitErr *err;
gint pgmNumber = 1, user=100, nAIPS = 0, nFITS=0;

/* Initialize Obit */
err = newObitErr();  /* Create error/message structure*/
mySystem = ObitSystemStartup ("test", pgmNumber, user, nAIPS, NULL,
                              nFITS, NULL, (oint)TRUE, (oint)FALSE, err);
ObitErrLog(err); /* show any error messages on err */
```

Here the nAIPS and nFITS give the number of explicit AIPS of FITS directories. The NULL following these arguments in the call sequence indicate that the standard defaults are to be used. If the directory name array passed is NULL, startup will search for standard environment names. These are $DA01, $DA02 ... for AIPS and $FITS, $FITS01 ... for FITS. If given explicitly, the directories can be either absolute or relative. If the directory number for FITS files is 0, the name is assumed to be an adequate relative or absolute pathname. Also specified in the ObitSystemStartup call are the name of the program (here "test") and a program number, pgmNumber, which is the POPS number for AIPS usage.

Obit classes also need initialization . This is done automatically for a class and all its parents whenever an Obit object is created.

Obit programs may generate scratch files which are recorded by the ObitSystem class. If these are not explicitly deleted (using the relevant Unref function) then they will be deleted by the ObitSystem shutdown:

```
/* Shutdown Obit */
mySystem = ObitSystemShutdown (mySystem);
```

3.1.2 Memory

Use and misuse of memory is one of the biggest source of bugs in the c family of languages. Obit uses the glib memory allocation routines to avoid some of the spectacular failures of the c malloc/free routines. As an additional layer of security and as an aid in finding memory leaks, Obit contains an ObitMem class for allocation and maintaining blocks of memory. This class is used extensively in Obit and maintains a list of optionally named blocks of memory allocated. To be deallocated, the pointer must be in the currently active list. This list can be printed using ObitMemPrint. The class can also be queried if a given pointer is inside an allocated block.

Many memory related problems (leaks, array overruns, NULL allocations...) can also be diagnosed using the memwatch package (as modified for glib/Obit) from Johan Lindh (http://www.linkdata.se/) This option is invoked using the -DMEMWATCH compile option and compiling and linking all Obit code. Very useful diagnostics are written into the file memwatch.log in the current working directory. This option overrides the ObitMem class usage. Don't forget to recompile/link without this option when through debugging as it incurs substantial overhead. It is important that all software linked together is all compiled with or without the -DMEMWATCH option. The Unix utility valgrind is very good (if very slow) at finding memory related problems.

3.1.3 Messages and error handling

Messages and error handling is by means of the ObitErr class. An ObitErr has an error condition flag and a stack of messages each with a category, i.e. error, informative... Low level routines enter

messages on the stack which is then explicitly logged and cleared using the ObitErrLog function. If an error is encountered, this condition is entered into the ObitErr structure and can be dealt with as appropriate by the calling routine. On entry, routines should check if an error condition exists and if so return. Tracebacks which are useful for debugging are obtained using the Obit_traceback macros.

Message logging can be customized by an application specific logging handler. Currently all messages are written to stdout or a specified log file.

The most useful ways of entering information into an ObitErr are through macroes defined in the class:

- Obit_log_error(err,errCode,format...)
  log error message.
- Obit_return_if_fail(expr,err,format...)
  Macro to evaluate expression, log in err and return on failure.
- Obit_retval_if_fail(expr,err,out,format...)
  Macro to evaluate expression, log in err and return a value on failure.
- Obit_traceback_msg(err,me,name)
  Macro for traceback when an error in a called routine is encountered. Writes traceback info and returns (no return value). Gives location (file, routine, line) where message generated.
- Obit_traceback_val(err,me,name,out)
  Macro for traceback logging with a return value Writes traceback info and returns, passing a value. Gives location (file, routine, line) where message generated.
- Obit_cfitsio(err)_error
  Macro to dump cfitsio error stack to an ObitErr.

The traceback macroes should be used to check for errors in routines called which cannot be handled in the current level and need to be passed to higher levels.

3.2 Obit Classes

Many, but not all, of the structures in Obit are inherited from the basal Obit virtual class. Obit includes template versions (ObitTEMPLATE*) files to assist in generating new classes; these template files contain the basic parts needed for a new class. The Obit class system is described below.

3.2.1 Organization

The class name of the class being defined in a file begins with the name of the class and include the following files for class xxx.

1. include/xxx.h
   Class interface definition. Defines structures, macroes and public functions.
2. include/xxxDef.h
   File included in include/xxx.h which defines the object structure. It first includes the corresponding file from its parent class.

3. include/xxxClassDef.h
   File included in include/xxx.h which defines the class structure. It first includes the corresponding file from its parent class.

4. src/xxx.c
   File containing the source code for the class; including class initialization.

Derived classes begin with the name of the parent class such that the inheritance is given in the class name.

Obit objects are connected to their respective classes by a pointer to a global class structure which contains a pointer to the parent class structure and function pointers for the class. Class member functions may be invoked either explicitly or via the class function pointers. Class objects, and all parent classes are initialized upon the instantiation of the first object of that class.

Obit objects have explicit creators but no explicit destructors. Instead, a referencing/defererencing system is used to destroy objects when the last reference to them is removed. A reference (pointer) is passed from the ObitRef (or class specific) function which increments the objects reference count. A reference is deleted using the ObitUnref (or class specific) function which decrements the object reference count, destroying it if it goes to zero and returns a null pointer. **The integrity of Obit depends on strict adherence to this convention.**

There are also "secret" references to objects needed to avoid a cyclical set of references. These references should not be used to manage the object's reference count, i.e. should not be passed to Ref or Unref functions. Local, convenience, copies of object pointers should also be treated this way.

The Obit virtual class object contains the following members:

1. gint32 ObitId;
   Recognition bit pattern to identify the start of an Obit object.

2. gpointer ClassInfo;
   ClassInfo pointer for class with name, base and function pointers.

3. gint ReferenceCount;
   Reference count for object (numbers of pointers attaching).

4. gchar *name;
   Name of object [Optional], this is used in error and informative messages.

The Obit virtual class structure contains the following members:

1. gboolean initialized;
   Have I been initialized?

2. gboolean hasScratch;
   Are disk resident "scratch" objects of this class possible?

3. gchar* ClassName;
   Name of class ("Obit")

4. gconstpointer ParentClass;
   Pointer to parent class ClassInfo, Null if none.

5. ObitClassInitFP ObitClassInit;
   Function pointer to Class initializer

6. newObitFP newObit;
   Function pointer to newObit (default constructor). Initializes class and parent if needed.
7. ObitCopyFP ObitCopy;
   Function pointer to shallow copy constructor.
8. ObitCloneFP ObitClone;
   Function pointer to deep copy constructor
9. ObitRefFP ObitRef;
   Function pointer to Object Reference
10. ObitUnrefFP ObitUnref;
    Function pointer to Object Unreference
11. ObitIsAFP ObitIsA;
    Function pointer to test if a class member.
12. ObitClearFP ObitClear;
    Private Function pointer to deallocation function.
13. ObitInitFP ObitInit;
    Private Function pointer to object initializer.

3.2.2 Inheritance

The object and class strucures are defined in the *Def.h and *ClassDef.h files as described above. Inheritance is by means of including the class object and class structure definition files (*Def.h and *ClassDef.h) at the beginning of the corresponding class files. This recursive include assures the full inherited structure is present in each object and class.

A template class, ObitTEMPLATE, is provided to simplify creating new classes. To create a new class copy the ObitTEMPLATE*.h and ObitTEMPLATE*.h to new files where TEMPLATE is replaced by the actual class name. Then, inside each of these files make the same substitutions. If the parent class is not the basic Obit class, then make the following changes:

1. Obit*.c;
   Near the top of the file change the valus of ObitParentGetClass to the Obit?GetClass function of the parent class.
2. Obit*.h;
   Add an include to the Obit*.h file of the parent class.
3. Obit*Def.h;
   Change include of ObitDef.h to the corresponding parent file, e.g. ObitTEMPLATEDef.h
4. Obit*ClassDef.h;
   Change include of ObitClassDef.h to the corresponding parent file, e.g. ObitTEMPLATE-ClassDef.h

3.2.3 Constructors

Obit objects are explicitly created by a constructor. The default constructor has the name new[ObitClassname] and accepts an optional name for the object. This name is used in error and informative messages and should reflect its usage. Derived classes may have other constructors.

3.2.4 Reference/destructors

Obit classes have no explicit destructors. Instead, this is handled by the Obit reference scheme. Pointers to an object are assigned by means of the ObitRef (or class specific) method which increments the object reference count. The ObitUnref (or class specific) method removes a reference, decrementing the reference count and destroying the object when the count goes to zero.

3.2.5 ObitInfoList

Most Obit classes have an ObitInfoList member which is a container for labeled arrays of information. Most control information is passed to and stored by the object in this member.

3.3 Obit Interface to Persistent Data

Obit supports multiple disk-resident forms; currently AIPS and FITS images and UV data as binary tables. Interfaces to external data are via derivations from the ObitIO class. Explicit derived classes are needed for each data type (UV Image, table...) and for each form (AIPS, FITS). Except for associating an object with an explicit external source (data file), most of Obit should not be aware of the source of the data outside of the explicit IO classes. The cfitsio package is used for basic FITS I/O. [1]

Directories containing AIPS or FITS data files are registered with Obit in the ObitSystem-Startup call. These directories are then referenced by a (1–rel) index.

3.3.1 Images

Images are accessed a plane (or row) at a time and are internally kept as a 1-D array of float. An ObitImage may have associated tables. A descriptor object (class ObitImageDesc) is used to describe the data. An ObitIO object is used to access the external data.

3.3.2 UV data

UV data is accessed as blocks of "rows" of visibility data which are internally kept as 1-D arrays of float for efficient access. The structure of UV data assumed is that of AIPS; "Random", descriptive data followed by a rectangular data array. "Compressed" data is supported. UV data in FITS format use binary tables for the visibility data but these tables are handled differently from normal tables for increased performance. An ObitUV may have associated tables. A descriptor object (class ObitUVDesc) is used to describe the data. An ObitIO object is used to access the external data.

3.3.3 Tables

ObitTables are always associated with Image, UV data or other data. A "row" in a table corresponds to a row in a FITS binary table and consists of an array of arrays of data of the same native type. Access is a row at a time and row data are converted to/from explicit class structures. A descriptor object (class ObitTableDesc) is used to describe the data. An ObitIO object is used to access the external data.

[1] The fitsio library is a package of Fortran and c language callable routines and is available at http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html. This package takes care of most of the details of reading and writing FITS files, including binary tables, and is well documented.

3.4 Threading

Multi–threaded executation in Obit is by means of the ObitThread class which uses the gthreads implementation of glib. Multi–threading is currently only supported for in–memory operations as cfitsio is not thread-safe. This class provides pools of threads to reduce the overhead of starting/stopping many threads doing the same operation. Multiple copies of a routine can be executed in parallel using ObitThreadIterator; the function to be run in multiple copies should have a signature of type ObitThreadFunc, e.g.:

```
gpointer ThreadFunction (gpointer arg);
```

A simple demo program is the following:

```
/* Demo threading concepts */
#include "Obit.h"
#include "ObitThread.h"
/* Function argument structure */
typedef struct {
  ObitThread *thread;  /* Threading control object
                          MUST be the same as passed to
                          ObitThreadIterator */
  olong      ithread;  /* Thread number */
  olong      ndata;    /* Number of entries in data */
  ofloat     *data;    /* data array */
} FuncArg;

/**
 * Demo routine to run in parallel threads.
 * Fills data values in array arg->data
 * Arguments are given in the FuncArg structure passed as arg
 * \param arg  Pointer to FuncArg argument with elements:
 * \li thread  Thread control object
 * \li ithread thread number
 * \li ndata   number of entries in data
 * \li data    data array
 * \return NULL
 */
gpointer fn(gpointer arg) {
  FuncArg  *farg = (FuncArg*)arg;
  olong j;

  /* Say I'm alive */
  fprintf (stdout," Running thread %d\n", farg->ithread);

  /* Fill data array */
  for (j=0; j<farg->ndata; j++) farg->data[j] = 0.01*j*farg->ithread;

  /* Indicate completion */
  ObitThreadPoolDone (farg->thread, (gpointer)&farg->ithread);
```

```
 return NULL;
} /* end fn */

/* Main program to demo threading */
int main ( int argc, char **argv )
{
  ObitThread *th=NULL;
  olong i, nThreads = 5;
  ObitThreadFunc func=(ObitThreadFunc)fn;
  gpointer *fargs[5];
  FuncArg args[5];

  /* Enable threading on 5 processors/cores */
  th = newObitThread();
  ObitThreadAllowThreads(th, nThreads);

  /* Initialize thread function argments */
  for (i=0; i<nThreads; i++) {
    args[i].thread  = th;
    args[i].ithread = i;
    args[i].ndata   = 1000;
    args[i].data    = g_malloc0(1000*sizeof(ofloat));
    fargs[i] = (gpointer)&args[i];
  }

  /* Execute - run 5 parallel copies of routine fn,
     will return on completion of all threads */
  ObitThreadIterator (th, nThreads, func, fargs);

  /* Shut down the thread pool */
  ObitThreadPoolFree (th);

  return 0;
} /* end main */
```

Synchronization of threads uses glib asynchronous message queues. Each copy of a threaded routine MUST call ObitThreadPoolDone to indicate it is completed (although there is a 1 minute timeout).

The ObitThread class provides mutexes for absolute locking of associated resources as well as RWLocks to allow multiple read accesses but only a single write (and no concurrent reads). Usage of threading needs OBIT_THREADS_ENABLED defined at compile time and the output of pkg-config –libs gthread-2.0 added to the libraries.

Multi-threading in a task using a user–specified parameter ("nThreads") given in the ObitInfoList myInput, is initialized by a call to ObitThreadInit (myInput). Alternatively, ObitThreadAllowThreads can be used as in the example above.

3.5 Documentation

Class documentation uses the doxygen system which uses specially coded comments in the code to generate documentation. The class documentation in html format starts at doc/doxygen/html/index.html.

3.6 History

Storing processing history is done principally via the ObitHistory class. Both FITS (as Tables) and AIPS cataloged data are supported. This class is derived from the Obit class. The AIPS conventions for history records are to give the program name followed by parameters in keyword=value form and to preceed any non parsable text by the FITS comment header comment delimiter '/'.

History tables are accessed as tables although the AIPS implementation is a pre-table version. History records are blocked into 70 character fixed strings althought AIPS internally uses 72.

History records are stored in a system dependent fashion. AIPS history records are stored in an AIPS HI* file with 72 characters per record (Obit only uses 70). For FITS files, history records are normally kept in A History binary table but can be read or written to the more traditional HISTORY keywords using ObitHistoryCopyHeader, or ObitHistoryCopy2Header for entire collections or ObitFileFITS functions ObitFileFITSReadHistory and ObitFileFITSWriteHistory for individual records. Access to the history component of an object (e.g. Image, UV data) can be obtained using the ObitInfoList containing the information defining the underlying file. This uses routine newObitHistoryValue. Then history lines can be read or written one at a time using ObitHistoryOpen, ObitHistoryReadRec, ObitHistoryWriteRec, ObitHistoryTimeStamp, ObitHistoryClose. The contents of entire history files may be copied using ObitHistoryCopy, or ObitHistoryCopyHeader to copy HISTORY records from a FITS header or ObitHistoryCopy2Header to copy a history file to a FITS header.

4 Interferometry Software

Much Obit software is designed to support radio interferometers. The ObitUV class represents an interface to radio interferometric data. The following sections describe some of the major classes for interferometry. Many of the algorithms used are adopted from AIPS.

4.1 UV Data Modeling

Lists of models or images can be Fourier transformed by the ObitSkyModel class and either divided into or subtracted from (added to) an ObitUV. Both DFT and Gridded model calculations are included for non spatially variant SkyModels and only DFT for these. Specialized subclasses of ObitSkyModel include:

- **ObitSkyModelVM**
  Base class of spatially variant sky models.
- **ObitSkyModelVMBeam**
  Spatial variation given by tabulated beam shapes.
- **ObitSkyModelVMIon**
  Spatial variation given by ionospheric model.
- **ObitSkyModelVMSquint**
  Spatial variation given by calculation of (E)VLA beam squint.

- **ObitSkyModelMF**
  Tabulated frequency dependent sky model.

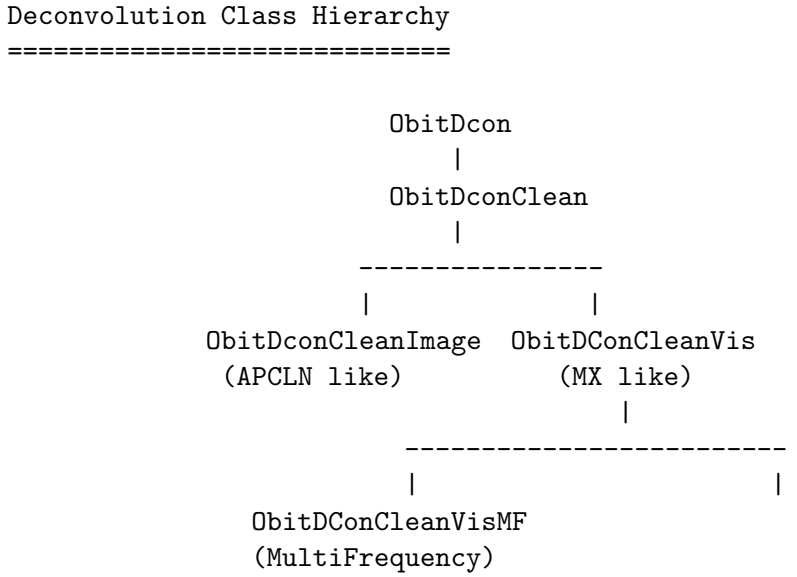
4.2 Imaging

The basic interface to the imaging software is the ObitUVImager class. Given an ObitUV with control parameters, this class can produce an ObitImageMosaic, an array of images that tile a region of the sky. Optionally, the array can be flattened into a single image. Specialized subclasses of ObitUVImager include:

- **ObitUVImagerIon**
  Imaging for spatial variation given by ionospheric model.
- **ObitUVImagerSquint**
  Imaging for spatial variation given by calculation of (E)VLA beam squint.
- **ObitUVImagerMF**
  Imaging for Tabulated frequency dependent sky model.
- **ObitUVImagerWB**
  Sault-Wieringa imaging.

4.3 Deconvolution

There is a heirarchy of classes derived from the ObitDCon (Deconvolution) virtual base class. All current implementations are variants of the CLEAN algorithm. Both Clark and SDI/Greisen CLEANS are implemented. This heirarchy is currently (Mar. 2010) only partially implemented but the high level design is shown in the following:



These classes are described in the following:

- ObitDCon
  Virtual base class of all deconvolution methods.

```
Members:
  ObitImageMosaic

Functions:
  Create, Copy, Clean, Deconvolve
  Set Image Mosaic
```

- ObitDConClean
  Virtual base class of CLEAN although much functionality used by derived classes is defined here.

```
Major Members:
   ObitDConCleanWindowList
   ObitDConCleanBmHist
   ObitDConCleanPxHist
   ObitDConCleanPixelList
   BeamPatch as FArray
   CLEAN control parameters (as arrays per field):
      Gain, Factor, Flux, Niter

Functions: (Called from Deconvolve)
   ObitDConCleanRestore     Restore subtracted components
   ObitDConCleanXRestore    Restores components from one field
                            appearing in another
   ObitDConCleanFlatten     Flatten multiple facets to one
   ObitDConCleanSelect      Select components to be subtracted
                            Load PixelList, BeamPatch, minor CLEANing
   ObitDConCleanSub         Subtract components, produce new residual
                            image(s),
   ObitDConCleanPixelStats  Prepare for minor cycle:
      ObitDConCleanPixelHist   Get image pixel histogram (one or more images)
      ObitDConCleanBeamHist    Get beam histogram
      ObitDConCleanDecide      Set Patch, min flux for next clean
```

- ObitDConCleanImage
  APCLN-like image only CLEAN. This implements a variation on the basic BGC CLEAN. This implementation is not limited to interferometry but can work on any image for which a stationary convolution ("Dirty Beam") can be derived.

```
Major Members:
   Transfer image (FFT of beam) as ObitCArray
   FFT object(s)

Functions:
   derived versions of:
```

```
   ObitDConCleanSub         Subtract components, produce new residual
                            Use convolution.  Uses Fourier Convolution
    Theorem to convolve components with the
    dirty beam to subtract from the previous
    residual image.
                            image(s)
```

- ObitDConCleanVis
  MX-like Cotton-Schwab CLEAN where components are subtracted from the visibility data which is reimaged to get the next round of residual images. SDI/Greisen variant also supported. Supports AIPS 2D or 3D type.

```
  Major Members:
     ObitUVImager

  Functions:
     ObitDConCleanVisPickNext   Selects next field to be cleaned

     derived versions of:
     ObitDConCleanDeconvolve  Control deconvolution algorithm.
     ObitDConCleanSub         Subtract components, produce new residual
                              Uses ObitSkyModel and ObitImager
```

- ObitDConCleanVisMR
  Multi-resolution CLEAN. NYI

A number of related classes are described in the following:

- ObitDConCleanWindow
  List of Windows for each field that describe the regions on which the CLEAN algorithm is to work. This list is expandable and hides the details of the window description by returning an image mask of the selected values. Initial implementation is for restangular and round windows.

- ObitDConCleanBmHist
  This class manages the beam histogram used in the B. G. Clark CLEAN algorithm which is used in Obit classes.

- ObitDConCleanPxHist
  This class manages the image pixel histogram used in the B. G. Clark CLEAN algorithm which is used in Obit classes.

- ObitDConCleanPxList
  This class manages lists of selected residuals from one or more images and implements the basic BGC CLEAN algorithm. Subclass ObitDConCleanPxListMF used for tabuilated spectrum wideband imaging.

5 Building Obit

Obit comes with a configure script to construct the Makefiles to build Obit. An installation script, InstallObit.sh, is included in the distribution which attempts to do a full installation. The following describes the installation of Obit in case InstallObit.sh does not work. Note: there are a number of third party packages that Obit requires and these must be installed prior to Obit. See the README file for more current details. The basic installation is thus:

```
% gtar xzvf Obit1.0.tgz
% cd Obit
% ./configure
% make
```

Several of the external packages are not critical and Obit will build without them. In particular, these are plplot, cfitsio, GSL, and FFTW3. However, if one of these is not installed or not found, then an attempt to use them will cause the program to abort. pgplot is used only in debugging and is not required for production use. The configure script searches in the standard places for external libraries and if they are in nonstandard places it will need some help. This help is provided in the form of arguments to the configure script:

```
  --with-x                use the X Window System (pgplot)
  --with-pgplot=DIR       search for PGPLOT in DIR
                          The libraries and header files are expected
                          here
  --with-cfitsio=DIR      search for CFITSIO in DIR/include and DIR/lib
  --with-cfitsio-includes=DIR
                          search for CFITSIO includes in DIR
  --with-fftw=DIR         search for FFTW in DIR/include and DIR/lib
  --with-fftw-includes=DIR
                          search for FFTW includes in DIR
  --with-gsl=DIR          search for GSL in DIR/include and DIR/lib
  --with-gsl-includes=DIR search for GSL includes in DIR
  --with-xmlrpc=DIR       search for XMLRPC in DIR/include and DIR/lib
  --with-xmlrpc-includes=DIR
                          search for XMLRPC includes in DIR
  --with-www=DIR          search for WWW in DIR/include and DIR/lib
  --with-www-includes=DIR search for WWW includes in DIR
  --with-zlib=DIR         search for ZLIB in DIR/include and DIR/lib
  --with-zlib-includes=DIR
                          search for ZLIB includes in DIR
```

The following gives an example of use when libraries are installed in $HOME/opt/:

```
 % configure --with-pgplot=$HOME/opt/pgplot \
--with-cfitsio=$HOME/opt/cfitsio \
--with-fftw=$HOME/opt/fftw
```

When using Obit from python, specify the PYTHONPATH environment variable as "Obit/python" where for Obit substitute the base directory of the Obit package.

5.1 Changing the Structure of Obit

Changes in the structure of the package may require modifications to the configure script. This requires a number of stages:

- aclocal
  This program generates an aclocal.m4 file as needed by autoconf. This is based on the configure.ac script and the files in the subdirectory m4 (needs -I m4 option) This utility is available from http://ftp.gnu.org. This needs the following files
  - configure.ac
    This files defines which libraries and features are needed and which Makefiles should be constructed
  - m4/cfitsio.m4
    This file contains macroes used in finding the cfitsio headers and library.
  - m4/fftw.m4
    This file contains macroes used in finding the FFTW headers and library.
  - m4/pgplot.m4
    This file contains macroes used in finding the pgplot headers and library.
- autoconf
  This utility is available from http://ftp.gnu.org. This program generates the configure script and needs the following
  - configure.ac
    This files defines which libraries and features are needed and which Makefiles should be constructed
  - aclocal.m4
    Generated by aclocal and contains the m4 macroes needed by autoconf
- configure
  This program (the configure script) generates the Makefiles specified in configure.ac based on the Makefile.in template files. This operation is largely by string substitution of @xxx@ type symbols. The values substituted are mostly determined by the m4 macroes in the *.m4 files. Configure uses the following:
  - missing
    Common stub for a few missing GNU programs while installing.
  - config.guess
    Attempt to guess a canonical system name.
  - config.sub
    Configuration validation subroutine script.
  - install-sh
    install - install a program, script, or datafile.
  - mkinstalldirs
    ??? Doesn't seem to be needed.
  - py-compile
    ?? Doesn't seem to be needed.

6 External software

Obit uses the following external packages:

- glib
  The gnu library for large scale software in ansci c. Available at http://www.gtk.org/.
- doxygen
  System for generating human readable software documentation from comments in the software. Available at http://www.stack.nl/~ dimitri/doxygen/ .
- cfitsio
  Basic FITS I/O package. Available at http://heasarc.gsfc.nasa.gov/docs/software/fitsio/fitsio.html.
- FFTW3
  Fast Fourier Transform. If configure does not find the FFTW3 library, FFTs are done using the GSL library if that was found. Available at http://www.fftw.org/. FFTW2 can also be used but it MUST be installed with the configure option –enable-float to be used by Obit.
- GSL
  GNU Scientific Library. If configure does not find the GSL library, the functions using it are stubbed such that the software will compile and link but any attempt to use a function relying on gsl routines will fail. Available at http://www.gnu.org/software/gsl/.
- Python
  Scripting language to which Obit has a binding. The binding interface is built using swig. (Note: use version 1.1? of swig. Although the output of swig is distributed so you don't need swig unless you want to modify the python interface. (http://www.swig.org/)) Python is available from http://www.python.org/download/. Python 2.3 or later is needed, 2.5 currently preferred.
- xmlrpc
  Interprocess communications is via xmlrpc using the xmlrpc-c and libwww libraries. xmlrpc-c-1.2 (http://download.sourceforge.net/xmlrpc-c/xmlrpc-c-%{ver}.tar.gz), w3c-libwww-5.4.0 (http://www.w3.org/Library/Distribution/w3c-libwww-%{ver}.tar.gz) libcurl works much better than libwww.
- zlib
  Probably everywhere you want to be. (if not, http://www.gzip.org/zlib/)

7 User interfaces

Currently only AIPS programs, user written c programs and python.

7.1 AIPS interface

A Fortran callable interface for Obit is defined in utility classes ObitAIPSFortran and ObitAIPSObject. Obit is only accessable from AIPS (POPS) in the form of tasks. For an example, see the OBTST task in subdirectory AIPS.

7.2 Python Interface

The intent of the Python interface is to allow scripting use of Obit. As such, the details of most data structures are not directly accessable from python but are manipulated at the file level by calling c routines. However, in order to allow proper scripting, some access to data members, especially the various descriptor classes is necessary. The contents of the various descriptors (e.g. ImageDesc, UVDesc) can be read and nonstructural values modified using an interface with a python dictionary object as an intermediary. Very rudimentary access is also provided for image pixel values and row data in tables. Images are accessed through FArray (arrays of floats) and many high level functions are available on FArrays.

The python interface is defined in the python subdirectory. The swig utility is used to interface the c library to python. The swig interface is defined in the ObitTypeMaps.swig (defines interface types) and *.inc which defines a python callable interface. The ObitTypeMaps.swig and *.inc files are concatenated by the Makefile before running swig. This generates a single, large shared library module to be imported into python. However, this is necessary to get all the classes into the same address space as Obit is dependent on class structures defining function pointers. A more elegant solution may be possible.

Note about swig. The maintainers of swig have modified the interface in more recent versions (after version 1.1?) so the output of swig, python/Obit_wrap.c, is distributed with other source code. The Make procedure will not attempt to run swig unless you've mofidied the interface (*.inc, *.swig).

The python interface defines a python class for each visible Obit class in a separate *.py file. The python/Obit classes are thin objects with basically a pointer to the c structure (the "me" member) but this allows mapping the python memory usage scheme onto Obit's similar but less automatic scheme. Thus the c objects created are automatically deleted when the python reference count drops to zero or the explicit destructor (del) is invoked. However, this operation in python is only performed during its occasional garbage collection; to ensure timely deallocation of large Obit objects use the explicit Unref function. All functions are implemented as nonclass member routines which take class arguments but many common functions (e.g. Open, Close, Read) are also implemented as class member functions. Class .py files are internally documented.

The convention for the routine names on the c side of the swig interface is the same as the Obit name but droping the initial "Obit". This adds another layer of routine call but allows translating the data types across the python/c divide. Swig (plus the type maps in swig and .inc files) helps but sometimes this is insufficient; having a uniform way of dealing with the interface helps. Some of the functions defined in the *.inc files correspond to macro expansions or straight access to member values in c.

On the python side of the interface, a separate layer of routines is added although generally the swig defined c routines are avaliable and while more efficient, this may circumvent the automatic destruction feature.

Several of the python modules define functions that combine multiple basic Obit function into higher level functional units (closer to the equivalent of c programs). The more complicated of the higher level routines (e.g. UVImager.py member UVImageInput = imaging component of AIPS IMAGR) have input structures containing defaults and which can be viewed using the class input function; this is similar in functionality to the AIPS "INPUT". Routines defined in the *.py files should contain the necessary information in the python doc strings.

7.3 Obit usage from Python

Obit can be accessed from python scripts or interactively. A number of scripts are available in the python subdirectory with names "script*.py". Either interactively or in a script, Obit needs to be initialized before usage:

```
import OTF, OErr, OSystem, Image
err=OErr.OErr()
userid = 100
nAIPS = -1
nFITS = 1
ObitSys=OSystem.OSystem  ("Python", 1, userid, nAIPS, ["Def"],
nFITS, ["./"], 1, 0, err)
OErr.printErrMsg(err, "Error with Obit startup")
```

where nAIPS are the number of AIPS directories specified, nFITS are the number of FITS directories specified. If either of these values are -1 (and "directory" = "Def") then the startup will search for standard environment names. These are $DA01, $DA02 ... for AIPS and $FITS,$FITS01 ... for FITS. If given explicitly, the directories can be either absolute or relative. If the directory number for FITS files is 0, the name is assumed to be an adequate relative or absolute pathname. In the example above, all standard AIPS directories will be used by default and the current directory for FITS files.

Functions in all *.py files should have documentation strings that can be accessed interactively as in the following example from UVImager. Note: help also works on entire python modules.

```
>>>  help(UVImager.UVImage)
Help on function UVImage in module UVImager:

UVImage(err, input={'Channel': 0, 'DoBeam': True, 'DoWeight': False, 'InData': None, 'OutImage
s': None, 'Robust': 0.0, 'TimeRange': [0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0], 'UVTaper': [0.
0, 0.0], 'WtBox': 0, 'WtFunc': 1, ...})
    Image a uv data set.

    UV Data is weighted and imaged producing an array of images.
    err     = Python Obit Error/message stack
    input   = input parameter dictionary

    Input dictionary entries:
    InData   = Input Python OTF to image
    OutImages= Output image mosaic, image objects should be previously defined
    DoBeam   = True if beams are to be made
    DoWeight = If True apply uniform weighting corrections
    Robust   = Briggs robust parameter. (AIPS definition)
    UVTaper  = UV plane taper, sigma in klambda as [u,v]
    WtSize   = Size of weighting grid in cells [same as image nx]
    WtBox    = Size of weighting box in cells [def 1]
    WtFunc   = Weighting convolution function [def. 1]
               1=Pill box, 2=linear, 3=exponential, 4=Gaussian
               if positive, function is of radius, negative in u and v.
```

```
    WtPower  = Power to raise weights to.  [def = 1.0]
               Note: a power of 0.0 sets all the output weights to 1 as modified
               by uniform/Tapering weighting.
               Applied in determing weights as well as after.
    Channel  = Channel (1-rel) number to image, 0-> all.
```

Many of the higher level functions have inputs in the form of a python dictionary whose current values can be displayed by the class input function, e.g.:

```
>>> UVImager.input(UVImager.UVImageInput)
Inputs for  UVImage
   InData  =  None  :  Input UV data
   OutImages  =  None  :  Output image mosaic
   DoBeam  =  True  :  True if beams are to be made
   DoWeight  =  False  :  If True apply uniform weighting corrections to uvdata
   Robust  =  0.0  :  Briggs robust parameter. (AIPS definition)
   UVTaper  =  [0.0, 0.0]  :  UV plane taper, sigma in klambda as [u,v]
   WtSize  =  -1  :  Size of weighting grid in cells [image]
   WtBox  =  0  :  Size of weighting box in cells [def 0]
   WtFunc  =  1  :  Weighting convolution function [def. 1]
   WtPower  =  1.0  :  Power to raise weights to.  [def = 1.0]
   Channel  =  0  :  Channel (1-rel) number to image, 0-> all.
Note: the values displayed for Obit objects are the names you give
them.
```

Obit messages and error handling is by means of the Obit type ObitErr which is an argument to most Obit routines. This contains informative as well as error messages; its contents can be displayed using e.g.:

```
>>> OErr.printErr(err)
** Message: information  : Hogbom CLEANed 300 components with 4.025047 Jy
** Message: information  : Reached minimum flux density -0.335160 Jy
** Message: information  : Scaling residuals by 1.079005
** Message: information  : Restoring 300 components
```

If the err object indicates an error, OErr.printErr(err), will raise an exception.

If scratch files are created (see OTF.ResidCal for an example) then an Obit shutdown will delete them:

```
# Shutdown Obit
OErr.printErr(err)
OSystem.Shutdown(ObitSys)
```

Note: AIPS files as well as FITS files can be accessed from python. However, OTF files have no equivalent in AIPS. An obscure problem of debugging Obit software running in python is getting the debuger to stop python when it knows about the Obit c code. The function Obit.Bomb() causes an abort (segmentation violation) which allows you to set breakpoints and restart.

8 Python script example

A python script that is the functional equivalent of the AIPS task HGEOM (as originally intended, interpolate the pixels of one image onto the grid defined by another) is shown in the following. The image is interpolated into a scratch image which is then copied to the output image as a quantized (1/4 RMS of noise) image , the old history is copied and new history records written.

```
# python/Obit equivalent of AIPSish HGEOM

import Obit, Image, ImageUtil, OSystem, OErr

# Init Obit
err=OErr.OErr()

# Use default directories
user = 100
pgmNumber = 1
ObitSys=OSystem.OSystem ("HGeom", pgmNumber, user, -1, ["Def"],
-1, ["Def"], 1, 0, err)
#  print any error messages and halt
OErr.printErrMsg(err, "Error with Obit startup")

# Define files (FITS)
# Image to be interpolated onto the grid of tmplFile
inDisk = 1
inFile   = 'input.fits'
# Image defining the output grid
tmplDisk = 1
tmplFile = 'template.fits'
# output file, the '!' allows overwriting an existing file
outDisk = 1
outFile  = '!HGeomOut.fits'

# Create Python/Obit objects attached to the data files
inImage   = Image.newPFImage("Input image",    inFile,   inDisk,   1, err)
tmplImage = Image.newPFImage("Template image", tmplFile, tmplDisk, 1, err)
outImage  = Image.newPFImage("Output image",   outFile,  outDisk,  0, err)
Image.PClone(tmplImage, outImage, err)   # Same structure etc.
OErr.printErrMsg(err, "Error initializing")

# Generate scratch file from tmplFile
tmpImage  = Image.PScratch(tmplImage, err)
tmpImage.Open(Image.WRITEONLY, err)   # Open
OErr.printErrMsg(err, "Error cloning template")

# Interpolate pixels to temporary file
ImageUtil.PInterpolateImage(inImage, tmpImage, err)
```

```
OErr.printErrMsg(err, "Error interpolating")

# Do history to scratch image as table
inHistory  = History.History("history", inImage.List, err)
outHistory = History.History("history", tmpImage.List, err)
History.PCopyHeader(inHistory, outHistory, err)
# Add this programs history
outHistory.Open(History.READWRITE, err)
outHistory.TimeStamp(" Start Obit "+ObitSys.pgmName,err)
outHistory.WriteRec(-1,ObitSys.pgmName+" / input = "+inFile,err)
outHistory.WriteRec(-1,ObitSys.pgmName+" / template = "+tmplFile,err)
outHistory.Close(err)
OErr.printErrMsg(err, "Error with history")

# Copy to quantized integer image with history
print "Write output image"
inHistory  = History.History("history", tmpImage.List, err)
Image.PCopyQuantizeFITS (tmpImage, outImage, err, inHistory=inHistory)

# Say what happened
print "Interpolated",inFile,"to",outFile,"a clone of ",tmplFile

# Shutdown Obit
OErr.printErr(err)  # Print any remaining Obit messages
OSystem.Shutdown(ObitSys)
```

9 C Language Example

The template Obit program test/Template.c is both a template to use for writing new Obit c programs and as an example which computes the mean and RMS of a window in an image. This program reads its input from a structured text file and writes its output into a similar file. In addition, many parameters can be passed on the command line. The program and its inputs are described in the test/Template.doc (ascii, NOT MSWord) file. The following example is the main routine in this program which shows the major features of an Obit program.

```
/* Program globals */
gchar *pgmName = "TEMPLATE";       /* Program name */
gchar *infile  = "TEMPLATE.inp";   /* File with program inputs */
gchar *outfile = "TEMPLATE.out";   /* File to contain program outputs */
gint  pgmNumber;       /* Program number (like POPS no.) */
gint  AIPSuser;        /* AIPS user number number (like POPS no.) */
gint  nAIPS=0;         /* Number of AIPS directories */
gchar **AIPSdirs=NULL; /* List of AIPS data directories */
gint  nFITS=0;         /* Number of FITS directories */
gchar **FITSdirs=NULL; /* List of FITS data directories */

int main ( int argc, char **argv )
```

```
/*----------------------------------------------------------------------- */
/*   Template Obit program - compute mean and RMS of an image             */
/*----------------------------------------------------------------------- */
{
  oint ierr = 0;
  ObitInfoList *myInput = NULL, *myOutput = NULL;
  ObitSystem   *mySystem= NULL;
  ObitImage    *inImage= NULL;
  ObitInfoType type;
  ObitErr      *err= NULL;
  gint32       dim[MAXINFOELEMDIM] = {1,1,1,1,1};
  gint         blc[IM_MAXDIM] = {1,1,1,1,1,1,1};
  gint         trc[IM_MAXDIM] = {0,0,0,0,0,0,0};
  gint         i, Aseq, disk, cno;
  gchar        *strTemp, inFile[128];
  gchar        Aname[13], Aclass[7], *Atype = "MA";
  gfloat       mean, rms;

   /* Startup - parse command line */
  err = newObitErr();
  myInput = TEMPLATEIn (argc, argv, err);
  if (err->error) ierr = 1;
  ObitErrLog(err); /* show any error messages on err */
  if (ierr!=0) return ierr;

  /* Initialize Obit */
  mySystem = ObitSystemStartup (pgmName, pgmNumber, AIPSuser, nAIPS, AIPSdirs,
                                nFITS, FITSdirs, (oint)TRUE, (oint)FALSE, err);
  if (err->error) ierr = 1;
  ObitErrLog(err); /* show any error messages on err */
  if (ierr!=0) return ierr;

  /* Create basic input Image Object */
  inImage = newObitImage("input Image");

  /* Get input parameters from myInput */
  ObitInfoListGet(myInput, "blc", &type, dim, blc, err); /* BLC */
  ObitInfoListGet(myInput, "trc", &type, dim, blc, err); /* TRC */

  /* File type  - could be either AIPS or FITS */
  ObitInfoListGetP (myInput, "Type", &type, dim, (gpointer)&strTemp);
  if (!strncmp (strTemp, "AIPS", 4)) { /* AIPS input */
    /* input AIPS disk */
    ObitInfoListGet(myInput, "inDisk", &type, dim, &disk, err);
    /* input AIPS name */
    ObitInfoListGet(myInput, "inName", &type, dim, Aname, err);
    /* input AIPS class */
    ObitInfoListGet(myInput, "inClass", &type, dim, Aclass, err);
```

```
  /* input AIPS sequence */
  ObitInfoListGet(myInput, "inSeq", &type, dim, &Aseq, err);

  /* Find catalog number */
  cno = ObitAIPSDirFindCNO(disk, AIPSuser, Aname, Aclass, Atype, Aseq, err);

  /* define image */
  ObitImageSetAIPS (inImage, OBIT_IO_byPlane, disk, cno, AIPSuser, blc, trc, err);

} else if (!strncmp (strTemp, "FITS", 4)) {  /* FITS input */
  /* input FITS file name */
  for (i=0; i<128; i++) inFile[i] = 0;
  ObitInfoListGet(myInput, "inFile", &type, dim, inFile, err);

  /* input FITS disk */
  ObitInfoListGet(myInput, "inDisk", &type, dim, &disk, err);

  /* define image */
  ObitImageSetFITS (inImage, OBIT_IO_byPlane, disk, inFile, blc, trc, err);

} else { /* Unknown type - barf and bail */
  Obit_log_error(err, OBIT_Error, "%s: Unknown Image type %s",
                 pgmName, strTemp);
  if (err->error) ierr = 1;
  ObitErrLog(err); /* show error messages */
  return ierr;
}

/* error check */
if (err->error) ierr = 1;
ObitErrLog(err); /* show any error messages on err */
if (ierr!=0) return ierr;

/* Open and read Image, image on member image, an ObitFArray */
ObitImageOpen (inImage, OBIT_IO_ReadOnly, err);
ObitImageRead (inImage, NULL, err);
if (err->error) ierr = 1;
ObitErrLog(err); /* show error messages */
if (ierr!=0) return ierr;

/* Get statistics from inImage FArray */
mean = ObitFArrayMean(inImage->image);
rms  = ObitFArrayRMS(inImage->image);

ObitImageClose (inImage, err); /* Close */
if (err->error) ierr = 1;
ObitErrLog(err); /* show error messages */
if (ierr!=0) return ierr;
```

```
  /* Tell results */
   Obit_log_error(err, OBIT_InfoErr,
  "%s: mean %f RMS %f", pgmName, mean, rms);

  /* Set up output */
  myOutput = defaultOutputs(err);
  dim[0] = 1; dim[1] = 1;
  ObitInfoListPut (myOutput, "mean", OBIT_float, dim, &mean, err);
  ObitInfoListPut (myOutput, "rms", OBIT_float, dim, &rms, err);
  ObitReturnDump (outfile, myOutput, err);

  /* show any messages and errors */
  if (err->error) ierr = 1;
  ObitErrLog(err);
  if (ierr!=0) return ierr;

  /* cleanup */
  myInput   = ObitInfoListUnref(myInput);    /* delete input list */
  myOutput  = ObitInfoListUnref(myOutput);   /* delete output list */
  inImage   = ObitUnref(inImage);

  /* Shutdown Obit */
  mySystem = ObitSystemShutdown (mySystem);

  return ierr;
} /* end of main */
```

10 Obit Tables

This document uses latex macroes which are translated by a perl script (ObitTables.pl) into the c source code.

10.1 LaTeX Macros

Tables used in Obit are defined in this document by using LaTeX macroes to formally define the table. These macroes are:

- tabletitle{Title of table, e.g. "Antenna table for uv data"}
- tablename{Name of table, e.g. "AN"}
- tableintro{Short description of class}
- tableover{Overview of usage of class}
- tablekey[{name}{type code}{ software name} {default value} {(range of indices)} {description}] Defines Table keyword, entries with a default value are not required to be present.

- tablecol[{name}{units}{type code} {(dimensionality)} {software name} {description}]
  Defines Table column. If the description contains the string "[OPTIONAL]" its presence will not be required; otherwise it will be checked.

11 Class ObitTableHistory

ObitTableHistory Class

11.1 Processing History for non-AIPS data

Table name: History

11.1.1 Introduction

[This class contains tabular data and allows access. This file is used in NON-AIPS applications to store the processing history of the associated data. Entries consist of 70 character strings which should be self labeling.]

11.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. The details of the storage in the buffer are kept in the ObitTableDesc. No Keywords in table.

11.1.3 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"ENTRY " : Type A** " " (70)
  Variable name root: entry
  Processing History entry

11.1.4 Modification History

1. W. D. Cotton 04/06/2004
   Revision 1: FOB

12 Class ObitTableAN

ObitTableAN Class

12.1 Antenna table for uv data

Table name: AN

12.1.1 Introduction

[This class contains tabular data and allows access. "AIPS AN" contains information about the locations and characteristics of antennas in a UV data set. Also time information and the state of the Earth's orientation. Each subarray in a uv data set will have it's own AN table in version numbers the same order as the subarray. Polarization calibration information is included.]

12.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

12.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 10, Range:
  Revision number of the table definition.
- **"NO_IF ": Type: J**, Variable name: numIF,
  Default value: 1, Range: ()
  The number of IFs
- **"ARRAYX ": Type: D**, Variable name: ArrayX,
  Default value: 0.0, Range:
  Array center X coord. (meters, earth center)
- **"ARRAYY ": Type: D**, Variable name: ArrayY,
  Default value: 0.0, Range:
  Array center Y coord. (meters, earth center)
- **"ARRAYZ ": Type: D**, Variable name: ArrayZ,
  Default value: 0.0, Range:
  Array center Z coord. (meters, earth center)

- **"GSTIA0 ": Type: D**, Variable name: GSTiat0,
  Default value: 0.0, Range:
  GST at time=0 (degrees) on the reference date
- **"DEGPDY ": Type: D**, Variable name: DegDay,
  Default value: 360.0, Range:
  Earth rotation rate (deg/IAT day)
- **"FREQ ": Type: D**, Variable name: Freq,
  Default value: 1.0, Range:
  Obs. Reference Frequency for subarray(Hz)
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYYMMDD", Range:
  Reference date as "YYYYMMDD"
- **"POLARX ": Type: E**, Variable name: PolarX,
  Default value: 0.0, Range:
  Polar position X (meters) on ref. date
- **"POLARY ": Type: E**, Variable name: PolarY,
  Default value: 0.0, Range:
  Polar position Y (meters) on ref. date
- **"UT1UTC ": Type: E**, Variable name: ut1Utc,
  Default value: , Range:
  UT1-UTC (time sec.)
- **"DATUTC ": Type: E**, Variable name: dataUtc,
  Default value: 0.0, Range:
  data time-UTC (time sec.)
- **"TIMSYS": Type: A**, Variable name: TimeSys,
  Default value: "IAT", Range:
  Time system, 'IAT' or 'UTC'
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"XYZHAND": Type: A**, Variable name: XYZHand,
  Default value: "RIGHT", Range:
  Handedness of coordinate system'
- **"FRAME": Type: A**, Variable name: FRAME,
  Default value: "UNKNOWN ", Range:
  Reference frame of coordinate system'
- **"NUMORB ": Type: J**, Variable name: numOrb,
  Default value: 0, Range: ()
  Number of orbital parameters

- **"NOPCAL ": Type: J**, Variable name: numPCal,
  Default value: 4, Range: ()
  Number of polarization calibration constants per IF
- **"FREQID ": Type: J**, Variable name: FreqID,
  Default value: 0, Range:
  Denotes the FQ ID for which the AN poln. parms have been modified.
- **"IATUTC ": Type: E**, Variable name: iatUtc,
  Default value: 0.0, Range:
  IAT - UTC (sec).
- **"POLTYPE ": Type: A**, Variable name: polType,
  Default value: " ", Range:
  Polarization parameterazation type, 'APPR', 'RAPPR', 'ORI-ELP'
- **"P_REFANT": Type: J**, Variable name: P_Refant,
  Default value: 0, Range:
  Polarization reference antenna
- **"P_DIFF01": Type: E**, Variable name: P_Diff01,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 1
- **"P_DIFF02": Type: E**, Variable name: P_Diff02,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 2
- **"P_DIFF03": Type: E**, Variable name: P_Diff03,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 3
- **"P_DIFF04": Type: E**, Variable name: P_Diff04,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 4
- **"P_DIFF05": Type: E**, Variable name: P_Diff05,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 5
- **"P_DIFF06": Type: E**, Variable name: P_Diff06,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 6
- **"P_DIFF07": Type: E**, Variable name: P_Diff07,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 7
- **"P_DIFF08": Type: E**, Variable name: P_Diff08,
  Default value: 0.0, Range:
  Right-Left Phase difference in radians IF 8

12.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"ANNAME " : Type A** " " (8)
  Variable name root: AntName
  Station name
- **"STABXYZ " : Type D** "METERS " (3)
  Variable name root: StaXYZ
  X,Y,Z offset from array center
- **"ORBPARM " : Type D** " " (numOrb)
  Variable name root: OrbParm
  Orbital parameters.
- **"NOSTA " : Type J** " " (1)
  Variable name root: noSta
  Station number, used as an index in other tables, uv data
- **"MNTSTA " : Type J** " " (1)
  Variable name root: mntSta
  Mount type, 0=altaz, 1=equatorial, 2=orbiting
- **"STAXOF " : Type E** "METERS " (1)
  Variable name root: staXof
  Axis offset
- **"DIAMETER" : Type E** "METERS " (1)
  Variable name root: diameter
  [OPTIONAL] Diameter of dish
- **"BEAMFWHM" : Type E** "DEG/M " (numIF)
  Variable name root: BeamFWHM
  [OPTIONAL] FWHM (degrees/meter) of single dish each IF, scales as wavelength
- **"POLTYA " : Type A** " " (4)
  Variable name root: polTypeA
  Feed A feed poln. type 'R','L','X','Y', actually only one valid character.
- **"POLAA " : Type E** "DEGREES " (1)
  Variable name root: PolAngA
  Feed A feed position angle
- **"POLCALA" : Type E** " " (numPCal,numIF)
  Variable name root: PolCalA
  Feed A poln. cal parameter.

- **"POLTYB " : Type A** " " (4)
  Variable name root: polTypeB
  Feed B feed poln. type 'R','L','X','Y'
- **"POLAB " : Type E** "DEGREES " (1)
  Variable name root: PolAngB
  Feed B feed position angle
- **"POLCALB" : Type E** " " (numPCal,numIF)
  Variable name root: PolCalB
  Feed B poln. cal parameter

12.1.5 Modification History

1. W. D. Cotton 02/03/2003
   Revision 1: Copied from AIPS
2. W. D. Cotton 28/07/2010
   Add keywords NO_IF, XYZHAND, FRAME, columns DIAMETER, BEAMFWHM, redefine numPCal

13 Class ObitTableAT

ObitTableAT Class

13.1 Antenna polarization table for uv data

Table name: AT

13.1.1 Introduction

[This class contains tabular data and allows access. "AIPS AT" contains Antenna polarization and something else. An ObitTableAT is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

13.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

13.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV": Type: J**, Variable name: revision,
  Default value: 3, Range:
  Revision number of the table definition.
- **"OBSCODE": Type: A**, Variable name: obscode,
  Default value: "AA000", Range:
  Observation code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYYMMDD"
- **"NO_STKD ": Type: J**, Variable name: numStkd,
  Default value: , Range:
  The number of Stokes(?)
- **"STK_1": Type: J**, Variable name: stk1,
  Default value: , Range:
  First Stokes(?)

- **"NO_BAND ": Type: J**, Variable name: numBand,
  Default value: , Range: ()
  The number of Bands(?).
- **"NO_CHAN": Type: J**, Variable name: numChan,
  Default value: , Range:
  The number of spectral channels.
- **"REF_FREQ": Type: D**, Variable name: refFreq,
  Default value: , Range:
  Reference Frequency.
- **"CHAN_BW ": Type: E**, Variable name: chanBW,
  Default value: , Range:
  Channel bandwidth.
- **"REF_PIXL ": Type: E**, Variable name: refPixl,
  Default value: , Range:
  Reference Pixel..
- **"NOPCAL": Type: J**, Variable name: noPCal,
  Default value: , Range:
  Number of polarization calibration parameters.
- **"POLTYPE": Type: A**, Variable name: polType,
  Default value: "APPROX ", Range:
  The Polarization calibration type.

13.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"TIME_INTERVAL " : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval of record
- **"ANNAME " : Type A** " " (8)
  Variable name root: AntName
  Station name
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number

- **"ARRAY " : Type J** " " (1)
  Variable name root: Array
  Array number
- **"FREQID ": Type: J**, Variable name: FreqID,
  Default value: 0, Range:
  The frequency ID.
- **"NO_LEVELS" : Type J** " " (1)
  Variable name root: numLevels
  Number of levels of something(?)
- **"POLTYA " : Type A** " " (1)
  Variable name root: polTypeA
  Feed A feed poln. type 'R','L','X','Y', actually only one valid character.
- **"POLAA " : Type E** "DEGREES " (numBand)
  Variable name root: PolAngA
  Feed A feed position angle
- **"POLCALA" : Type E** " " (numBand)
  Variable name root: PolCalA
  Feed A poln. cal parameter.
- **"POLTYB " : Type A** " " (1)
  Variable name root: polTypeB
  Feed B feed poln. type 'R','L','X','Y'
- **"POLAB " : Type E** "DEGREES " (numBand)
  Variable name root: PolAngB
  Feed B feed position angle
- **"POLCALB" : Type E** " " (numBand)
  Variable name root: PolCalB
  Feed B poln. cal parameter

13.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

14 Class ObitTableBL

ObitTableBL Class

14.1 Baseline dependent calibration for uv data

Table name: BL

14.1.1 Introduction

[This class contains tabular data and allows access. "AIPS BL" contains baseline dependent additive and multiplicative terms for the correction of UV data. Each table row contains is for a single baseline and has a correction for each IF. An ObitTableBL is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

14.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

14.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"NO_ANT ": Type: J**, Variable name: numAnt,
  Default value: , Range:
  The number of antennas.
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations.
- **"NO_IF ": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs.

14.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X**=bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.

The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type E** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"SOURCE ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number
- **"ANTENNA1 " : Type J** " " (1)
  Variable name root: ant1
  First antenna number of baseline
- **"ANTENNA2 " : Type J** " " (1)
  Variable name root: ant2
  Second antenna number of baseline
- **"FREQ ID " : Type J** " " (1)
  Variable name root: FreqID
  Freqid number
- **"REAL M#NO_POL" : Type E** " " (numIF)
  Variable name root: RealM
  Real (Multiplicative correction Poln # NO_POL)
- **"IMAG M#NO_POL" : Type E** " " (numIF)
  Variable name root: ImagM
  Imaginary (Multiplicative correction Poln # NO_POL)
- **"REAL A#NO_POL" : Type E** " " (numIF)
  Variable name root: RealA
  Real (Additive correction Poln # NO_POL)
- **"IMAG A#NO_POL" : Type E** " " (numIF)
  Variable name root: ImagA
  Imaginary (Additive correction Poln # NO_POL)

14.1.5 Modification History

1. W. D. Cotton 03/03/2003
   Revision 1: Copied from AIPS

15 Class ObitTableBP

ObitTableBP Class

15.1 UV data BandPass calibration table

Table name: BP

15.1.1 Introduction

[This class contains tabular data and allows access. "AIPS BP" contains bandpass calibration informatioin for UV data. An ObitTableBP is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

15.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

15.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"NO_ANT ": Type: J**, Variable name: numAnt,
  Default value: , Range:
  The number of antennas
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of antennas
- **"NO_IF ": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs
- **"NO_CHAN": Type: J**, Variable name: numChan,
  Default value: , Range: ()
  Number of frequency channels
- **"STRT_CHN": Type: J**, Variable name: startChan,
  Default value: 1, Range:
  Start channel number

- **"NO_SHFTS": Type: J**, Variable name: numShifts,
  Default value: 1, Range:
  If numShifts = 1 BP entries are from cross-power data, if 2 are from total power, if 3 are a mixture, anything else then type is unknown and will assume cross-power
- **"LOW_SHFT": Type: J**, Variable name: lowShift,
  Default value: 1, Range:
  Most negative shift
- **"SHFT_INC": Type: J**, Variable name: shiftInc,
  Default value: 1, Range:
  Shift increment
- **"BP_TYPE": Type: A**, Variable name: BPType,
  Default value: , Range:
  BP type: ' ' =¿ standard BP table, CHEBSHEV' =¿ Chebyshev polynomial coeff.

15.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"INTERVAL " : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval of record
- **"SOURCE ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number
- **"ANTENNA " : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"BANDWIDTH " : Type E** "HZ " (1)
  Variable name root: BW
  andwidth of an individual channel

- **"CHN_SHIFT " : Type D** " " (numIF)
  Variable name root: ChanShift
  Frequency shift for each IF
- **"FREQ ID " : Type J** " " (1)
  Variable name root: FreqID
  Freq. id number
- **"REFANT #NO_POL" : Type J** " " (1)
  Variable name root: RefAnt
  Reference Antenna
- **"WEIGHT #NO_POL" : Type E** " " (numIF)
  Variable name root: Weight
  Weights for complex bandpass
- **"REAL #NO_POL" : Type E** " " (numChan,numIF)
  Variable name root: Real
  Real (channel gain Poln # NO_POL)
- **"IMAG #NO_POL" : Type E** " " (numChan,numIF)
  Variable name root: Imag
  Imaginary (channel gain Poln # NO_POL)

15.1.5 Modification History

1. W. D. Cotton 03/03/2003
   Revision 1: Copied from AIPS

16 Class ObitTableCC

16.1 Clean Components table.

Table name: CC

16.1.1 Introduction

[This class contains tabular data and allows access. "AIPS CC" contains sky model components obtained either by CLEANing or fitting. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

16.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. The details of the storage in the buffer are kept in the ObitTableDesc. For non–point components, there are extra parameters:

1. param[0] = major axis size (deg)
2. param[1] = minor axis size (deg)
3. param[2] = position angle (deg)
4. param[3] = type
   - 0 = point,
   - 1 = Gaussian on sky,
   - 2 = Convolved Gaussian,
   - 3 = Uniform optically thin sphere
   - type + 10 => following parameters are spectral coefficients of powers of $ln(\nu/\nu_0)$
   - type + 20 => following parameters are tabulated spectra
5. param[4] spectral index (if type>10, <= 19)
6. param[5] spectral curvature (if type>10, <= 19)
7. param[6] spectral curvature square (if type>10, <= 19)
8. param[3+i] spectral channel i (1-rel) (if type>20, <= 29)

No Keywords in table.

16.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"NO_PARMS": Type: J**, Variable name: noParms,
  Default value: 0, Range: (0,5)
  Number of parameters for non point components, usually 0 or 4+

16.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X**=bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"FLUX " : Type E** "JY " (1)
  Variable name root: Flux
  Component flux density
- **"DELTAX " : Type E** "DEGREE " (1)
  Variable name root: DeltaX
  Component X position.
- **"DELTAY " : Type E** "DEGREE " (1)
  Variable name root: DeltaY
  Component Y position.
- **"PARMS " : Type E** " " (noParms)
  Variable name root: parms
  [OPTIONAL] Component parameters;

16.1.5 Modification History

1. W. D. Cotton 15/11/2004
   Revision 1: Copied from AIPS

17 Class ObitTableCL

17.1 CaLibration table for uv data.

Table name: CL

17.1.1 Introduction

[This class contains tabular data and allows access. "AIPS CL" contains amplitude/phase/delay and rate calibration information to be applied to a multi source UV data set. An ObitTableCL is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

17.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

17.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 10, Range:
  Revision number of the table definition
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations
- **"NO_IF ": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs
- **"NO_ANT ": Type: J**, Variable name: numAnt,
  Default value: , Range:
  The number of antennas in table
- **"NO_TERM": Type: J**, Variable name: numTerm,
  Default value: 0, Range: ()
  The number of terms in model polynomial

- **"MGMOD ": Type: D**, Variable name: mGMod,
  Default value: 1.0, Range:
  The Mean Gain modulus

17.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time of the solution
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  Solution interval.
- **"SOURCE ID" : Type J** " " (1)
  Variable name root: SourID
  Source Identifier number
- **"ANTENNA NO." : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"SUBARRAY" : Type J** " " (1)
  Variable name root: SubA
  Subarray number.
- **"FREQ ID" : Type J** " " (1)
  Variable name root: FreqID
  Frequency ID
- **"I.FAR.ROT" : Type E** "RAD/M**2" (1)
  Variable name root: IFR
  Ionospheric Faraday Rotation
- **"GEODELAY " : Type D** "SECONDS " (numTerm)
  Variable name root: GeoDelay
  Geometric delay polynomial series at TIME
- **"DOPPOFF " : Type E** "SEC/SEC " (numIF)
  Variable name root: DopplerOff
  Doppler offset for each IF
- **"ATMOS " : Type E** "SECONDS " (1)
  Variable name root: atmos
  Atmospheric delay

- **"DATMOS " : Type E** "SEC/SEC " (1)
  Variable name root: Datmos
  Time derivative of ATMOS
- **"MBDELAY#NO_POL" : Type E** "SECONDS " (1)
  Variable name root: MBDelay
  Multiband delay poln # NO_POL
- **"CLOCK #NO_POL" : Type E** "SECONDS " (1)
  Variable name root: clock
  "Clock" epoch error
- **"DCLOCK #NO_POL" : Type E** "SEC/SEC" (1)
  Variable name root: Dclock
  Time derivative of CLOCK
- **"DISP #NO_POL" : Type E** "SECONDS " (1)
  Variable name root: dispers
  Dispersive delay (sec at wavelength = 1m)for Poln # NO_POL
- **"DDISP #NO_POL" : Type E** "SEC/SEC" (1)
  Variable name root: Ddispers
  Time derivative of DISPfor Poln # NO_POL
- **"REAL#NO_POL" : Type E** " " (numIF)
  Variable name root: Real
  Real (gain Poln # NO_POL)
- **"IMAG#NO_POL" : Type E** " " (numIF)
  Variable name root: Imag
  Imaginary (gain Poln # NO_POL)
- **"RATE #NO_POL" : Type E** "SEC/SEC " (numIF)
  Variable name root: Rate
  Residual fringe rate Poln # NO_POL
- **"DELAY #NO_POL" : Type E** "SECONDS " (numIF)
  Variable name root: Delay
  Residual group delay Poln # NO_POL
- **"WEIGHT #NO_POL" : Type E** " " (numIF)
  Variable name root: Weight
  Weight of soln. Poln # NO_POL
- **"REFANT #NO_POL" : Type J** " " (numIF)
  Variable name root: RefAnt
  Reference antenna Poln # NO_POL

17.1.5 Modification History

1. W. D. Cotton 28/02/2003
   Revision 1: Copied from AIPS

18 Class ObitTableCQ

ObitTableCQ Class

18.1 VLBA Correlator parameter frequency tablefor uv data

Table name: CQ

18.1.1 Introduction

[This class contains tabular data and allows access. "AIPS CQ" table contains VLBA-like correlation parameters which are used for making a number of instrumental corrections. An ObitTableCQ is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

18.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

18.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV": Type: J**, Variable name: revision,
  Default value: 1, Range:
  Revision number of the table definition
- **"NO_IF": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs

18.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"FRQSEL " : Type J** " " (1)
  Variable name root: FrqSel
  Frequency ID {IFQDCQ in AIPSish}
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number {ISUBCQ}
- **"FFT_SIZE " : Type J** " " (numIF)
  Variable name root: FFTSize
  Size of FFT in correlator {NFFTCQ}
- **"NO_CHAN " : Type J** " " (numIF)
  Variable name root: numChan
  No. of channels in correlator{NCHCQ}
- **"SPEC_AVG " : Type J** " " (numIF)
  Variable name root: SpecAvg
  Spectral averaging factor{NSAVCQ}
- **"EDGE_FRQ " : Type D** "HZ " (numIF)
  Variable name root: EdgeFreq
  Edge frequency {DFRQCQ}
- **"CHAN_BW " : Type D** "HZ " (numIF)
  Variable name root: ChanBW
  Channel bandwidth {DCBWCQ}
- **"TAPER_FN " : Type A** " " (8,numIF)
  Variable name root: TaperFn
  Taper function {LTAPCQ}
- **"OVR_SAMP " : Type J** " " (numIF)
  Variable name root: OverSamp
  Oversampling factor {NOVSCQ}
- **"ZERO_PAD " : Type J** " " (numIF)
  Variable name root: ZeroPad
  Zero-padding factor {NZPDCQ}
- **"FILTER " : Type J** " " (numIF)
  Variable name root: Filter
  Filter type {IFLTCQ}
- **"TIME_AVG " : Type E** "SECONDS " (numIF)
  Variable name root: TimeAvg
  Time averaging interval {TAVGCQ}
- **"NO_BITS " : Type J** " " (numIF)
  Variable name root: numBits
  Quantization (no. of bits per recorded sample){NBITCQ}

- **"FFT_OVLP " : Type J** " " (numIF)
  Variable name root: FFTOverlap
  FFT overlap factor {IOVLCQ}

18.1.5 Modification History

1. W. D. Cotton 04/03/2003
   Revision 1: Copied from AIPS

19 Class ObitTableCT

ObitTableCT Class

19.1 Earth orientation parameter table for uv data

Table name: CT

19.1.1 Introduction

[This class contains tabular data and allows access. "AIPS CT" contains Earth orientation model parameters. An ObitTableCT is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

19.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

19.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV": Type: J**, Variable name: revision,
  Default value: 2, Range:
  Revision number of the table definition.
- **"OBSCODE": Type: A**, Variable name: obscode,
  Default value: "AA000", Range:
  Observation code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYYMMDD", Range:
  Reference date as "YYYYMMDD"
- **"NO_STKD ": Type: J**, Variable name: numStkd,
  Default value: , Range:
  The number of Stokes(?)
- **"STK_1": Type: J**, Variable name: stk1,
  Default value: , Range:
  First Stokes(?)

- **"NO_BAND ": Type: J**, Variable name: numBand,
  Default value: , Range: ()
  The number of Bands(?).
- **"NO_CHAN": Type: J**, Variable name: numChan,
  Default value: , Range:
  The number of spectral channels.
- **"REF_FREQ": Type: D**, Variable name: refFreq,
  Default value: , Range:
  Reference Frequency.
- **"CHAN_BW ": Type: E**, Variable name: chanBW,
  Default value: , Range:
  Channel bandwidth.
- **"REF_PIXL ": Type: E**, Variable name: refPixl,
  Default value: , Range:
  Reference Pixel..
- **"C_SRVR ": Type: A**, Variable name: CSrvr,
  Default value: , Range:
  Who knows?
- **"C_VERSN": Type: A**, Variable name: CVersn,
  Default value: , Range:
  Who knows?
- **"A_VERSN": Type: A**, Variable name: AVersn,
  Default value: , Range:
  Who knows?
- **"I_VERSN": Type: A**, Variable name: IVersn,
  Default value: , Range:
  Who knows?
- **"E_VERSN": Type: A**, Variable name: EVersn,
  Default value: , Range:
  Who knows?
- **"ACCELGRV": Type: D**, Variable name: accelgrv,
  Default value: , Range:
  Acceleration due to gravity m/s/s
- **"E-FLAT": Type: D**, Variable name: Eflat,
  Default value: , Range:
  Earth Flattening???
- **"EARTHRAD": Type: D**, Variable name: earthrad,
  Default value: , Range:
  Earth radius

- **"MMSEMS": Type: D**, Variable name: mmsems,
  Default value: , Range:
  ???
- **"EPHEPOC": Type: J**, Variable name: ephepoc,
  Default value: , Range:
  Ephemeris epoch???
- **"TIDELAG": Type: D**, Variable name: tidelag,
  Default value: , Range:
  TIDELAG
- **"GAUSS": Type: D**, Variable name: gauss,
  Default value: , Range:
  GAUSS
- **"GMMOON": Type: D**, Variable name: gmmoon,
  Default value: , Range:
  GMMOON
- **"GMSUN": Type: D**, Variable name: gmsun,
  Default value: , Range:
  GMSUN
- **"LOVE_H": Type: D**, Variable name: loveH,
  Default value: , Range:
  LOVE_H
- **"LOVE_L": Type: D**, Variable name: loveL,
  Default value: , Range:
  LOVE_L
- **"PRE_DATA": Type: D**, Variable name: preData,
  Default value: , Range:
  PRE_DATA
- **"REL_DATA": Type: D**, Variable name: relData,
  Default value: , Range:
  REL_DATA
- **"TIDALUT1": Type: J**, Variable name: tidalut1,
  Default value: , Range:
  TIDALUT1
- **"TSECAU": Type: D**, Variable name: tsecau,
  Default value: , Range:
  TSECAU
- **"U-GRV-CN ": Type: J**, Variable name: UGrvCn,
  Default value: , Range:
  U-GRV-CN

- **"VLIGHT ": Type: J**, Variable name: vlight,
  Default value: , Range:
  Speed of light m/s

19.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"UT1-UTC" : Type D** "SECONDS " (1)
  Variable name root: ut1utc
  UT1-UTC
- **"IAT-UTC" : Type D** "SECONDS " (1)
  Variable name root: iatutc
  IAT-UTC
- **"A1-UTC" : Type D** "SECONDS " (1)
  Variable name root: a1utc
  A1-UTC
- **"UT1 TYPE" : Type A** " " (1)
  Variable name root: ut1Type
  UT1 TYPE
- **"WOBXY " : Type D** "MILLIARC" (2)
  Variable name root: wobXY
  Earth pole wobble
- **"WOB TYPE" : Type A** " " (1)
  Variable name root: wobType
  WOB TYPES
- **"DPSI" : Type D** "RAD " (1)
  Variable name root: dpsi
  DPSI
- **"DDPSI" : Type D** "RAD/SEC " (1)
  Variable name root: ddpsi
  DDPSI
- **"DEPS" : Type D** "RAD " (1)
  Variable name root: deps
  DEPS

- **"DDEPS " : Type D** "RAD/SEC " (1)
  Variable name root: ddeps
  DDEPS

19.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

20 Class ObitTableFG

ObitTableFG Class

20.1 Flag table for uv data documentation

Table name: FG

20.1.1 Introduction

[This class contains tabular data and allows access. "AIPS FG" contains descriptions of data to be ignored An ObitTableFG is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

20.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc. No Keywords in table.

20.1.3 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"SOURCE " : Type J** " " (1)
  Variable name root: SourID
  Source ID as defined in the SOURCE table
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number
- **"FREQ ID " : Type J** " " (1)
  Variable name root: freqID
  Frequency ID number
- **"ANTS " : Type J** " " (2)
  Variable name root: ants
  first and secong antenna numbers for a baseline, 0=>all

- **"TIME RANGE " : Type E** "DAYS " (2)
  Variable name root: TimeRange
  Start and end time of data to be flagged
- **"IFS " : Type J** " " (2)
  Variable name root: ifs
  First and last IF numbers to flag
- **"CHANS " : Type J** " " (2)
  Variable name root: chans
  First and last channel numbers to flag
- **"PFLAGS " : Type X** " " (4)
  Variable name root: pFlags
  Polarization flags, same order as in data, T=>flagged
- **"REASON " : Type A** " " (24)
  Variable name root: reason
  Reason for flagging

20.1.4 Modification History

1. W. D. Cotton 02/03/2003
   Revision 1: Copied from AIPS

21 Class ObitTableFQ

21.1 UV data FreQuency Table

Table name: FQ

21.1.1 Introduction

[This class contains tabular data and allows access. "AIPS FQ" contains frequency related information for "IF" in uv data. An "IF" is a construct that allows sets of arbitrarily spaced frequencies. An ObitTableFQ is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

21.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

21.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"NO_IF ": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs, used to dimension table column entries

21.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"FRQSEL " : Type J** " " (1)
  Variable name root: fqid
  Frequency ID number for row, this is a random parameter in the uv data

- **"IF FREQ " : Type D** "HZ " (numIF)
  Variable name root: freqOff
  Offset from reference frequency for each IF

- **"CH WIDTH" : Type E** "HZ " (numIF)
  Variable name root: chWidth
  Bandwidth of an individual channel, now always written and read as a signed value
- **"TOTAL BANDWIDTH " : Type E** "HZ " (numIF)
  Variable name root: totBW
  Total bandwidth of the IF, now written and read as an unsigned value
- **"SIDEBAND " : Type J** " " (numIF)
  Variable name root: sideBand
  Sideband of the IF (-1 => lower, +1 => upper), now always written and read as +1

21.1.5 Modification History

1. W. D. Cotton 02/03/2002
   Revision 1: Copied from AIPS

22 Class ObitTableGC

ObitTableGC Class

22.1 Gain curve table for uv data

Table name: GC

22.1.1 Introduction

[This class contains tabular data and allows access. "AIPS GC" contains Antenna gain curve information An ObitTableGC is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

22.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

22.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"NO_BAND ": Type: J**, Variable name: numBand,
  Default value: , Range: ()
  The number of Bands(?).
- **"NO_POL": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations.
- **"NO_TABS": Type: J**, Variable name: numTabs,
  Default value: , Range: ()
  The number of ??.
- **"TABREV": Type: J**, Variable name: revision,
  Default value: 3, Range:
  Revision number of the table definition.

22.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubArray
  Subarray number
- **"FREQ ID " : Type J** " " (1)
  Variable name root: FreqID
  Frequency id of scan
- **"TYPE_#NO_POL" : Type J** " " (numBand)
  Variable name root: Type
  TYPE_#NO_POL
- **"NTERM_#NO_POL" : Type J** " " (numBand)
  Variable name root: NTerm
  NTERM_#NO_POL
- **"X_TYP_#NO_POL" : Type J** " " (numBand)
  Variable name root: XTyp
  X_TYP_#NO_POL
- **"Y_TYP_#NO_POL" : Type J** " " (numBand)
  Variable name root: YTyp
  Y_TYP_#NO_POL
- **"X_VAL_#NO_POL" : Type E** " " (numBand)
  Variable name root: XVal
  X_TYP_#NO_POL
- **"Y_VAL_#NO_POL" : Type E** " " (numTabs,numBand)
  Variable name root: YVal
  Y_TYP_#NO_POL
- **"GAIN_#NO_POL" : Type E** " " (numTabs,numBand)
  Variable name root: gain
  GAIN_#NO_POL
- **"SENS_#NO_POL" : Type E** "K/JY " (numBand)
  Variable name root: sens
  X_TYP_#NO_POL

22.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

23 Class ObitTableIM

ObitTableIM Class

23.1 IM table for uv data

Table name: IM

23.1.1 Introduction

[This class contains tabular data and allows access. "AIPS IM" contains the correlator model. An ObitTableIM is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

23.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

23.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYYMMDD"
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations.
- **"OBSCODE": Type: A**, Variable name: obscode,
  Default value: "AA000", Range:
  Observation code
- **"NO_STKD ": Type: J**, Variable name: numStkd,
  Default value: , Range:
  The number of Stokes(?)
- **"NO_BAND ": Type: J**, Variable name: numBand,
  Default value: , Range: ()
  The number of Bands(?).

- **"NO_CHAN": Type: J**, Variable name: numChan,
  Default value: , Range:
  The number of spectral channels.
- **"REF_FREQ": Type: D**, Variable name: refFreq,
  Default value: , Range:
  Reference Frequency.
- **"CHAN_BW ": Type: E**, Variable name: chanBW,
  Default value: , Range:
  Channel bandwidth.
- **"REF_PIXL ": Type: E**, Variable name: refPixl,
  Default value: , Range:
  Reference Pixel..
- **"NPOLY ": Type: J**, Variable name: npoly,
  Default value: , Range: ()
  The number of polynomial terms.
- **"TABREV": Type: J**, Variable name: tabrev,
  Default value: 2, Range:
  Revision number of the table definition.
- **"REVISION": Type: E**, Variable name: revision,
  Default value: 1.0, Range:
  Revision number of the table definition.

23.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"TIME_INTERVAL " : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval of record
- **"SOURCE_ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number

- **"ARRAY " : Type J** " " (1)
  Variable name root: Array
  Array number
- **"FREQID" : Type J** " " (1)
  Variable name root: FreqID
  Frequency ID
- **"I.FAR.ROT" : Type E** "RAD/M**2" (1)
  Variable name root: IFR
  Ionospheric Faraday Rotation
- **"FREQ.VAR" : Type E** "HZ " (numBand)
  Variable name root: FreqVar
  FREQ.VAR (?)
- **"PDELAY_#NO_POL " : Type D** "SECONDS " (npoly,numBand)
  Variable name root: PDelay
  Phase delay
- **"GDELAY_#NO_POL " : Type D** "SECONDS " (npoly)
  Variable name root: GDelay
  Group delay
- **"PRATE_#NO_POL " : Type D** "SECONDS " (npoly,numBand)
  Variable name root: PRate
  Phase delay rate
- **"GRATE_#NO_POL " : Type D** "SECONDS " (npoly)
  Variable name root: GRate
  Group delay rate
- **"DISP_#NO_POL " : Type E** "SECONDS " ()
  Variable name root: Disp
  Dispersion
- **"DDISP_#NO_POL " : Type E** "SEC/SEC " ()
  Variable name root: DRate
  Dispersion rate

23.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

24 Class ObitTableNI

ObitTableIN Class

24.1 Ionospheric calibration table class for uv data

Table name: NI

24.1.1 Introduction

[This class contains tabular data and allows access. "AIPS NI" contains the results of ionospheric modeling fits. The ionospheric phase screen over the array is modeled in terms of a Zernike polynomial. An ObitTableNI is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This was formerly the IN table but an AIPS bug does nasty things to "IN" tables. This class is derived from the ObitTable class.]

24.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

24.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 1, Range:
  Revision number of the table definition.
- **"NUM_COEF": Type: J**, Variable name: numCoef,
  Default value: 5, Range: ()
  Number of Zernike coefficients
- **"H_ION ": Type: E**, Variable name: heightIon,
  Default value: 1.0e10, Range:
  Height of the ionospheric phase screen (km?)
- **"REF_FREQ": Type: D**, Variable name: refFreq,
  Default value: , Range:
  Reference frequency for the phase screen model

24.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  ime interval of the solution
- **"ANTENNA NO." : Type J** " " (1)
  Variable name root: antNo
  Antenna number, 0=> all
- **"SOURCE ID" : Type J** " " (1)
  Variable name root: SourId
  Source number, 0=> all
- **"SUBARRAY" : Type J** " " (1)
  Variable name root: SubA
  Subarray number, 0=> all
- **"WEIGHT " : Type E** " " (1)
  Variable name root: weight
  Weight
- **"COEF " : Type E** " " (numCoef)
  Variable name root: coef
  Zernike model coeffients

24.1.5 Modification History

1. W. D. Cotton 28/02/2003
   Revision 1: Copied from AIPS

25 Class ObitTableMC

ObitTableMC Class

25.1 MC table for uv data

Table name: MC

25.1.1 Introduction

[This class contains tabular data and allows access. "AIPS MC" contains VLBA Correlator model calculations. An ObitTableMC is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

25.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

25.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"OBSCODE": Type: A**, Variable name: obscode,
  Default value: "AA000", Range:
  Observation code
- **"NO_POL": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations.
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYYMMDD", Range:
  Reference date as "YYYYMMDD"
- **"NO_STKD ": Type: J**, Variable name: numStkd,
  Default value: , Range:
  The number of Stokes(?)
- **"STK_1": Type: J**, Variable name: stk1,
  Default value: , Range:
  First Stokes(?)

- **"NO_BAND ": Type: J**, Variable name: numBand,
  Default value: , Range: ()
  The number of Bands(?).
- **"NO_CHAN": Type: J**, Variable name: numChan,
  Default value: , Range:
  The number of spectral channels.
- **"REF_FREQ": Type: D**, Variable name: refFreq,
  Default value: , Range:
  Reference Frequency.
- **"CHAN_BW ": Type: E**, Variable name: chanBW,
  Default value: , Range:
  Channel bandwidth.
- **"REF_PIXL ": Type: E**, Variable name: refPixl,
  Default value: , Range:
  Reference Pixel..
- **"FFT_SIZE": Type: J**, Variable name: FFTSize,
  Default value: , Range:
  FFT size.
- **"OVERSAMP": Type: J**, Variable name: oversamp,
  Default value: , Range:
  Oversampling factor
- **"ZERO_PAD": Type: J**, Variable name: zeroPad,
  Default value: , Range:
  Zero padding factor
- **"TAPER_FN": Type: A**, Variable name: taperFn,
  Default value: , Range:
  Tapering function
- **"TABREV": Type: J**, Variable name: revision,
  Default value: 1, Range:
  Revision number of the table definition.

25.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.

- **"SOURCE_ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number
- **"ARRAY " : Type J** " " (1)
  Variable name root: Array
  Array number
- **"FREQID" : Type J** " " (1)
  Variable name root: FreqID
  Frequency ID
- **"ATMOS " : Type D** "SECONDS " (1)
  Variable name root: atmos
  Atmospheric delay
- **"DATMOS " : Type D** "SEC/SEC " (1)
  Variable name root: Datmos
  Time derivative of ATMOS
- **"GDELAY" : Type D** "SECONDS " (1)
  Variable name root: GDelay
  Group delay
- **"GRATE " : Type D** "SECONDS " (1)
  Variable name root: GRate
  Group delay rate
- **"CLOCK_#NO_POL" : Type D** "SECONDS " (1)
  Variable name root: clock
  "Clock" epoch error
- **"DCLOCK_#NO_POL" : Type D** "SEC/SEC" (1)
  Variable name root: Dclock
  Time derivative of CLOCK
- **"LO_OFFSET_#NO_POL" : Type E** "HZ " (1)
  Variable name root: LOOffset
  LO Offset
- **"DLO_OFFSET_#NO_POL" : Type E** "Hz/SEC " (1)
  Variable name root: DLOOffset
  Time derivative of LO offset
- **"DISP #NO_POL" : Type E** "SECONDS " (1)
  Variable name root: disp
  Dispersive delay (sec at wavelength = 1m)for Poln # NO_POL

- **"DDISP #NO_POL" : Type E** "SEC/SEC" (1)
  Variable name root: Ddisp
  Time derivative of DISPfor Poln # NO_POL

25.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

26 Class ObitTableMF

ObitTableMF Class

26.1 AIPS Model Fit Table

Table name: MF

26.1.1 Introduction

[This class contains tabular data and allows access. "AIPS MF" the results of a model fit to an image or uv data of the type produced by SAD/VSAD. An ObitTableMF is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

26.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

26.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"DEPTH1": Type: E**, Variable name: depth1,
  Default value: 1, Range:
  Dimensions 3 in image
- **"DEPTH2": Type: E**, Variable name: depth2,
  Default value: 1, Range:
  Dimensions 4 in image
- **"DEPTH3": Type: E**, Variable name: depth3,
  Default value: 1, Range:
  Dimensions 5 in image1
- **"DEPTH4": Type: E**, Variable name: depth4,
  Default value: 1, Range:
  Dimensions 6 in image
- **"DEPTH5": Type: E**, Variable name: depth5,
  Default value: 1, Range:
  Dimensions 7 in image

26.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"PLANE " : Type E** " " (1)
  Variable name root: plane
  Plane number
- **"PEAK INT" : Type E** "JY/BEAM " (1)
  Variable name root: Peak
  Peak Ipol
- **"I FLUX " : Type E** "JY " (1)
  Variable name root: IFlux
  Integrated Ipol flux
- **"DELTAX " : Type E** "DEGREE " (1)
  Variable name root: DeltaX
  X offset of center
- **"DELTAY " : Type E** "DEGREE " (1)
  Variable name root: DeltaY
  Y offset of center
- **"MAJOR AX" : Type E** "DEGREE " (1)
  Variable name root: MajorAx
  Fitted major axis size
- **"MINOR AX" : Type E** "DEGREE " (1)
  Variable name root: MinorAx
  Fitted minor axis size
- **"POSANGLE" : Type E** "DEGREE " (1)
  Variable name root: PosAngle
  Fitted PA
- **"Q FLUX " : Type E** "JY " (1)
  Variable name root: QFlux
  Integrated Q flux density
- **"U FLUX " : Type E** "JY " (1)
  Variable name root: UFlux
  Integrated U flux density
- **"V FLUX " : Type E** "JY " (1)
  Variable name root: VFlux
  Integrated Y flux density

- **"ERR PEAK" : Type E** "JY/BEAM " (1)
  Variable name root: errPeak
  rror in Peak Ipol
- **"ERR FLUX" : Type E** "JY " (1)
  Variable name root: errIFlux
  rror in Integrated Ipol flux
- **"ERR DLTX" : Type E** "DEGREE " (1)
  Variable name root: errDeltaX
  Error in X offset of center
- **"ERR DLTY" : Type E** "DEGREE " (1)
  Variable name root: errDeltaY
  Error in Y offset of center
- **"ERR MAJA" : Type E** "DEGREE " (1)
  Variable name root: errMajorAx
  Error in Fitted major axis size
- **"ERR MINA" : Type E** "DEGREE " (1)
  Variable name root: errMinorAx
  Error in Fitted minor axis size
- **"ERR PA " : Type E** "DEGREE " (1)
  Variable name root: errPosAngle
  Error in Fitted PA
- **"ERR QFLX" : Type E** "JY " (1)
  Variable name root: errQFlux
  Error in Integrated Q flux density
- **"ERR UFLX" : Type E** "JY " (1)
  Variable name root: errUFlux
  Error in Integrated U flux density
- **"ERR VFLX" : Type E** "JY " (1)
  Variable name root: errVFlux
  Error in Integrated Y flux density
- **"TYPE MOD" : Type E** " " (1)
  Variable name root: TypeMod
  Model type 1 = Gaussian
- **"D0 MAJOR" : Type E** "DEGREE " (1)
  Variable name root: D0Major
  Deconvolved best major axis
- **"D0 MINOR" : Type E** "DEGREE " (1)
  Variable name root: D0Minor
  Deconvolved best minor axis

- **"D0 POSANG" : Type E** "DEGREE " (1)
  Variable name root: D0PosAngle
  Deconvolved best PA
- **"D- MAJOR" : Type E** "DEGREE " (1)
  Variable name root: DmMajor
  Deconvolved least major axis
- **"D- MINOR" : Type E** "DEGREE " (1)
  Variable name root: DmMinor
  econvolved least minor axis
- **"D- POSAN" : Type E** "DEGREE " (1)
  Variable name root: DmPosAngle
  Deconvolved least PA
- **"D+ MAJOR" : Type E** "DEGREE " (1)
  Variable name root: DpMajor
  Deconvolved most major axis
- **"D+ MINOR" : Type E** "DEGREE " (1)
  Variable name root: DpMinor
  Deconvolved most minor axis
- **"D+ POSAN" : Type E** "DEGREE " (1)
  Variable name root: DpPosAngle
  Deconvolved most PA
- **"RES RMS " : Type E** "JY/BEAM " (1)
  Variable name root: ResRMS
  RMS of Ipol residual
- **"RES PEAK" : Type E** "JY/BEAM " (1)
  Variable name root: ResPeak
  Peak in Ipol residual
- **"RES FLUX" : Type E** "JY " (1)
  Variable name root: ResFlux
  Integrated Ipol in residual
- **"CENTER X" : Type E** "PIXEL " (1)
  Variable name root: PixelCenterX
  Center x position in pixels
- **"CENTER Y" : Type E** "PIXEL " (1)
  Variable name root: PixelCenterY
  Center y position in pixels
- **"MAJ AXIS" : Type E** "PIXEL " (1)
  Variable name root: PixelMajorAxis
  Fitted major axis in pixels

- **"MIN AXIS" : Type E** "PIXEL " (1)
  Variable name root: PixelMinorAxis
  Fitted minor axis in pixels
- **"PIXEL PA" : Type E** "DEGREE " (1)
  Variable name root: PixelPosAngle
  Fitted PA(?)

26.1.5 Modification History

1. W. D. Cotton 03/03/2003
   Revision 1: Copied from AIPS

27 Class ObitTableNX

ObitTableNX Class

27.1 iNdeX table for uv data

Table name: NX

27.1.1 Introduction

[This class contains tabular data and allows access. "AIPS NX" contains an index for a uv data file giving the times, source and visibility range for a sequence of scans. A scan is a set of observations in the same mode and on the same source. An ObitTableNX is the front end to a persistent disk resident structure. This class is derived from the ObitTable class.]

27.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc. No Keywords in table.

27.1.3 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type E** "DAYS " (1)
  Variable name root: Time
  The center time of the sscan.
- **"TIME INTERVAL " : Type E** "DAYS " (1)
  Variable name root: TimeI
  Duration of scan
- **"SOURCE ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID as defined in then SOURCE table
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number

- **"START VIS " : Type J** " " (1)
  Variable name root: StartVis
  First visibility number (1-rel) in scan
- **"END VIS " : Type J** " " (1)
  Variable name root: EndVis
  Last visibility number (1-rel) in scan
- **"FREQ ID " : Type J** " " (1)
  Variable name root: FreqID
  Frequency id of scan

27.1.4 Modification History

1. W. D. Cotton 02/03/2003
   Revision 1: Copied from AIPS

28 Class ObitTableOB

ObitTableOB Class

28.1 Orbital antenna table for uv data

Table name: OB

28.1.1 Introduction

[This class contains tabular data and allows access. "AIPS OB" contains Orbiting VLBI station parameters. An ObitTableOB is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

28.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

28.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV": Type: J**, Variable name: revision,
  Default value: 2, Range:
  Revision number of the table definition.

28.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number

- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number
- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"ORBXYZ " : Type D** "METERS " (3)
  Variable name root: orbXYZ
  Obital position
- **"VELXYZ " : Type D** "M/SEC " (3)
  Variable name root: velXYZ
  Orbital velocity
- **"SUN_ANGLE " : Type E** "DEGREES " (3)
  Variable name root: sunAngle
  Relative position of the Sun
- **"ECLIPSE " : Type E** "DAYS " (4)
  Variable name root: eclipse
  ECLIPSE
- **"ORIENTATION " : Type E** "DEGREES " (1)
  Variable name root: orientation
  ORIENTATION

28.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

29 Class ObitTableOF

ObitTableOF Class

29.1 OF table for uv data

Table name: OF

29.1.1 Introduction

[This class contains tabular data and allows access. "AIPS OF" contains some sort of VLA operational information. An ObitTableOF is the front end to a persistent disk resident structure. This class is derived from the ObitTable class.]

29.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

29.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 10, Range:
  Revision number of the table definition
- **"OBSCODE": Type: A**, Variable name: obscode,
  Default value: "AA000", Range:
  Observation code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYYMMDD", Range:
  Reference date as "YYYYMMDD"

29.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable

name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type E** "DAYS " (1)
  Variable name root: Time
  The center time of the sscan.
- **"TIME INTERVAL " : Type E** "DAYS " (1)
  Variable name root: TimeI
  Duration of scan
- **"SOURCE ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID as defined in then SOURCE table
- **"ANTENNA NO." : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number
- **"FREQ ID " : Type J** " " (1)
  Variable name root: FreqID
  Frequency id of scan
- **"REFN PTG FLAG VA " : Type L** " " (1)
  Variable name root: refPnt
  Reference pointing(?) flag
- **"SHADOWED FLAG VA " : Type L** " " (1)
  Variable name root: shadow
  Antenna shadowed(?) flag

29.1.5 Modification History

1. W. D. Cotton 02/03/2003
   Revision 1: Copied from AIPS

30 Class ObitTablePC

ObitTablePC Class

30.1 Pulsed cal. table for uv data

Table name: PC

30.1.1 Introduction

[This class contains tabular data and allows access. "AIPS PC" contains pulsed phase cal. info An ObitTablePC is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

30.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

30.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV": Type: J**, Variable name: revision,
  Default value: 3, Range:
  Revision number of the table definition.
- **"NO_POL": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations.
- **"NO_BAND ": Type: J**, Variable name: numBand,
  Default value: , Range: ()
  The number of Bands(?).
- **"NO_TONES": Type: J**, Variable name: numTones,
  Default value: , Range: ()
  The number of Tones.

30.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"SOURCE_ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number
- **"ARRAY " : Type J** " " (1)
  Variable name root: Array
  Array number
- **"FREQID" : Type J** " " (1)
  Variable name root: FreqID
  Frequency ID
- **"CABLE_CAL" : Type D** "SECONDS " (1)
  Variable name root: CableCal
  CABLE_CAL
- **"STATE_#NO_POL " : Type E** "PERCENT " (4,numBand)
  Variable name root: State
  State counts(?)
- **"PC_FREQ_#NO_POL " : Type D** "HZ " (numTones,numBand)
  Variable name root: PCFreq
  Frequencies of the phase tones
- **"PC_REAL_#NO_POL " : Type E** " " (numTones,numBand)
  Variable name root: PCReal
  Real part of tone phase
- **"PC_IMAG_#NO_POL " : Type E** " " (numTones,numBand)
  Variable name root: PCImag
  Imaginary part of tone phase
- **"PC_RATE_#NO_POL " : Type E** "SEC/SEC " (numTones,numBand)
  Variable name root: PCRate
  Tone rate phase

30.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

31 Class ObitTablePS

ObitTablePS Class

31.1 Processing Summary

Table name: PS

31.1.1 Introduction

[This class contains tabular data and allows access. "AIPS PS" contains the summary of the processing of a source. An ObitTablePS is the front end to a persistent disk resident structure. This class is derived from the ObitTable class.]

31.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

31.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 1, Range:

31.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"FIELD NAME " : Type A** " " (16)
  Variable name root: FieldName
  Field name (source)

- **"OBSDATE " : Type A** " " (8)
  Variable name root: obsdate
  Observing ref. date.(YYYYMMDD)
- **"RA POINT " : Type D** "DEG " (1)
  Variable name root: RAPoint
  Pointing RA at epoch 2000
- **"DEC POINT" : Type D** "DEG " (1)
  Variable name root: DecPoint
  Pointing Dec at epoch 2000
- **"STATUS " : Type A** " " (8)
  Variable name root: Status
  Processing status
- **"IQU " : Type L** " " (3)
  Variable name root: IQU
  IQU Flags True if processed
- **"SPECIAL " : Type L** " " (1)
  Variable name root: Special
  Special processing applied
- **"NO. FIELDS" : Type J** " " (1)
  Variable name root: NoFields
  Number of sub fields
- **"CC LIMIT " : Type L** " " (3)
  Variable name root: CCLimit
  Flags to using all possible CC.
- **"PERCENT " : Type E** " " (3)
  Variable name root: PerCent
  Percent of visibility data I,Q,U
- **"PIX MAX " : Type E** " " (3)
  Variable name root: PixMax
  Pixel max (Jy) I,Q,U
- **"PIX MIN " : Type E** " " (3)
  Variable name root: PixMin
  Pixel min (Jy) I,Q,U
- **"QUALITY " : Type E** " " (3)
  Variable name root: Quality
  Quality measure, RMS I,Q,U
- **"COMMENTS " : Type A** " " (80)
  Variable name root: Comment
  Comments

31.1.5 Modification History

1. W. D. Cotton 03/15/2006
   Revision 1: Copied from AIPS

32 Class ObitTableSN

SolutioN table for UV data.

32.1 SolutioN table for UV data

Table name: SN

32.1.1 Introduction

[This class contains tabular data and allows access. "AIPS SN" amplitude/phase/delay and rate calibration information derived from a calibration solution and can be used either for "self calibration" or correction a multisource CaLibration table. An ObitTableSN is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

32.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

32.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 10, Range:
  Revision number of the table definition.
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations.
- **"NO_IF ": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs
- **"NO_ANT ": Type: J**, Variable name: numAnt,
  Default value: , Range:
  The number of antennas in table.
- **"NO_NODES": Type: J**, Variable name: numNodes,
  Default value: 0, Range:
  The number of interpolation nodes.

- **"MGMOD ": Type: D**, Variable name: mGMod,
  Default value: 1.0, Range:
  The Mean Gain modulus
- **"APPLIED": Type: L**, Variable name: isApplied,
  Default value: FALSE, Range:
  True if table has been applied to a CL table.

32.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X**=bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time of the solution
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  Solution interval.
- **"SOURCE ID" : Type J** " " (1)
  Variable name root: SourID
  Source Identifier number
- **"ANTENNA NO." : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"SUBARRAY" : Type J** " " (1)
  Variable name root: SubA
  Subarray number.
- **"FREQ ID" : Type J** " " (1)
  Variable name root: FreqID
  Frequency ID
- **"I.FAR.ROT" : Type E** "RAD/M**2" (1)
  Variable name root: IFR
  Ionospheric Faraday Rotation
- **"NODE NO." : Type J** " " (1)
  Variable name root: NodeNo
  Node number
- **"MBDELAY#NO_POL" : Type E** "SECONDS " (1)
  Variable name root: MBDelay
  Multiband delay poln # NO_POL

- **"REAL#NO_POL" : Type E** " " (numIF)
  Variable name root: Real
  Real (gain Poln # NO_POL)
- **"IMAG#NO_POL" : Type E** " " (numIF)
  Variable name root: Imag
  Imaginary (gain Poln # NO_POL)
- **"DELAY #NO_POL" : Type E** "SECONDS " (numIF)
  Variable name root: Delay
  Residual group delay Poln # NO_POL
- **"RATE #NO_POL" : Type E** "SEC/SEC " (numIF)
  Variable name root: Rate
  Residual fringe rate Poln # NO_POL
- **"WEIGHT #NO_POL" : Type E** " " (numIF)
  Variable name root: Weight
  Weight of soln. Poln # NO_POL
- **"REFANT #NO_POL" : Type J** " " (numIF)
  Variable name root: RefAnt
  Reference antenna Poln # NO_POL

32.1.5 Modification History

1. W. D. Cotton 28/02/2002
   Revision 1: Copied from AIPS

33 Class ObitTableSU

ObitTableSU Class

33.1 SoUrce table for uv data

Table name: SU

33.1.1 Introduction

[This class contains tabular data and allows access. "AIPS SU" contains information about astronomical sources. An ObitTableSU is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

33.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

33.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"NO_IF ": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs
- **"VELTYP ": Type: A**, Variable name: velType,
  Default value: "LSR", Range:
  Velocity type
- **"VELDEF ": Type: A**, Variable name: velDef,
  Default value: "RADIO", Range:
  Velocity definition 'RADIO' or 'OPTICAL'
- **"FREQID ": Type: J**, Variable name: FreqID,
  Default value: 0, Range:
  The Frequency ID for which the source parameters are relevant.

33.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"ID. NO. " : Type J** " " (1)
  Variable name root: SourID
  Source ID
- **"SOURCE " : Type A** " " (16)
  Variable name root: Source
  Source name
- **"QUAL " : Type J** " " (1)
  Variable name root: Qual
  Source qualifier
- **"CALCODE " : Type A** " " (4)
  Variable name root: CalCode
  Calibrator code
- **"IFLUX " : Type E** "JY " (numIF)
  Variable name root: IFlux
  Total Stokes I flux density per IF
- **"QFLUX " : Type E** "JY " (numIF)
  Variable name root: QFlux
  Total Stokes Q flux density per IF
- **"UFLUX " : Type E** "JY " (numIF)
  Variable name root: UFlux
  Total Stokes U flux density per IF
- **"VFLUX " : Type E** "JY " (numIF)
  Variable name root: VFlux
  Total Stokes V flux densityper IF
- **"FREQOFF " : Type D** "HZ " (numIF)
  Variable name root: FreqOff
  Frequency offset (Hz) from IF nominal per IF
- **"BANDWIDTH" : Type D** "HZ " (1)
  Variable name root: Bandwidth
  Bandwidth
- **"RAEPO " : Type D** "DEGREES " (1)
  Variable name root: RAMean
  Right ascension at mean EPOCH (actually equinox)

- **"DECEPO " : Type D** "DEGREES " (1)
  Variable name root: DecMean
  Declination at mean EPOCH (actually equinox)
- **"EPOCH " : Type D** "YEARS " (1)
  Variable name root: Epoch
  Mean Epoch (really equinox) for position in yr. since year 0.0
- **"RAAPP " : Type D** "DEGREES " (1)
  Variable name root: RAApp
  Apparent Right ascension
- **"DECAPP " : Type D** "DEGREES " (1)
  Variable name root: DecApp
  Apparent Declination
- **"LSRVEL " : Type D** "M/SEC " (numIF)
  Variable name root: LSRVel
  LSR velocity per IF
- **"RESTFREQ" : Type D** "HZ " (numIF)
  Variable name root: RestFreq
  Line rest frequency per IF
- **"PMRA " : Type D** "DEG/DAY " (1)
  Variable name root: PMRa
  Proper motion (deg/day) in RA
- **"PMDEC " : Type D** "DEG/DAY " (1)
  Variable name root: PMDec
  Proper motion (deg/day) in declination

33.1.5 Modification History

1. W. D. Cotton 09/03/2003
   Revision 1: Copied from AIPS

34 Class ObitTableTY

ObitTableTY Class

34.1 Temperature table for uv data

Table name: TY

34.1.1 Introduction

[This class contains tabular data and allows access. "AIPS TY" contains Antenna and system temperature information. An ObitTableTY is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

34.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

34.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"NO_IF": Type: J**, Variable name: numIF,
  Default value: , Range: ()
  The number of IFs
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations.
- **"REVISION": Type: J**, Variable name: revision,
  Default value: 10, Range:
  Revision number of the table definition.

34.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X**=bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable

name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type E** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"TIME INTERVAL " : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval of record
- **"SOURCE ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"ANTENNA NO." : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number
- **"SUBARRAY " : Type J** " " (1)
  Variable name root: SubA
  Subarray number
- **"FREQ ID " : Type J** " " (1)
  Variable name root: FreqID
  Frequency id of scan
- **"TSYS #NO_POL" : Type E** "KELVINS " (numIF)
  Variable name root: Tsys
  Real (gain Poln # NO_POL)
- **"TANT #NO_POL" : Type E** "KELVINS " (numIF)
  Variable name root: Tant
  Imaginary (gain Poln # NO_POL)

34.1.5 Modification History

1. W. D. Cotton 02/22/2006
   Revision 1: Copied from AIPS

35 Class ObitTableVL

ObitTableVL Class

35.1 NVSS full format catalog file

Table name: VL

35.1.1 Introduction

[This class contains tabular data and allows access. "AIPS VL" contains a catalog of sources in the format produced for the NVSS survey. The table contains an index in the INDEX?? keywords that give the first table row number (1-rel) for a given hour of RA. An ObitTableVL is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

35.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

35.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 1, Range:
  Revision number of the table definition
- **"BM_MAJOR": Type: E**, Variable name: BeamMajor,
  Default value: , Range:
  Restoring beam major axis in deg.
- **"BM_MINOR": Type: E**, Variable name: BeamMinor,
  Default value: , Range:
  Restoring beam minor axis in deg.
- **"BM_PA": Type: E**, Variable name: BeamPA,
  Default value: , Range:
  Restoring beam position angle of major axis in deg.
- **"SORTORT": Type: J**, Variable name: SortOrder,
  Default value: , Range:
  Column number for sort (neg -¿ descending)

- **"NUM_INDE": Type: J**, Variable name: numIndexed,
  Default value: 1, Range:
  Number of rows in table when indexed
- **"INDEX00": Type: J**, Variable name: index00,
  Default value: 1, Range:
  First entry for RA=00 h
- **"INDEX01": Type: J**, Variable name: index01,
  Default value: 1, Range:
  First entry for RA=01 h
- **"INDEX02": Type: J**, Variable name: index02,
  Default value: 1, Range:
  First entry for RA=02 h
- **"INDEX03": Type: J**, Variable name: index03,
  Default value: 1, Range:
  First entry for RA=03 h
- **"INDEX04": Type: J**, Variable name: index04,
  Default value: 1, Range:
  First entry for RA=04 h
- **"INDEX05": Type: J**, Variable name: index05,
  Default value: 1, Range:
  First entry for RA=05 h
- **"INDEX06": Type: J**, Variable name: index06,
  Default value: 1, Range:
  First entry for RA=06 h
- **"INDEX07": Type: J**, Variable name: index07,
  Default value: 1, Range:
  First entry for RA=07 h
- **"INDEX08": Type: J**, Variable name: index08,
  Default value: 1, Range:
  First entry for RA=08 h
- **"INDEX09": Type: J**, Variable name: index09,
  Default value: 1, Range:
  First entry for RA=09 h
- **"INDEX10": Type: J**, Variable name: index10,
  Default value: 1, Range:
  First entry for RA=10 h
- **"INDEX11": Type: J**, Variable name: index11,
  Default value: 1, Range:
  First entry for RA=11 h

- **"INDEX12": Type: J**, Variable name: index12,
  Default value: 1, Range:
  First entry for RA=12 h
- **"INDEX13": Type: J**, Variable name: index13,
  Default value: 1, Range:
  First entry for RA=13 h
- **"INDEX14": Type: J**, Variable name: index14,
  Default value: 1, Range:
  First entry for RA=14 h
- **"INDEX15": Type: J**, Variable name: index15,
  Default value: 1, Range:
  First entry for RA=15 h
- **"INDEX16": Type: J**, Variable name: index16,
  Default value: 1, Range:
  First entry for RA=16 h
- **"INDEX17": Type: J**, Variable name: index17,
  Default value: 1, Range:
  First entry for RA=17 h
- **"INDEX18": Type: J**, Variable name: index18,
  Default value: 1, Range:
  First entry for RA=18 h
- **"INDEX19": Type: J**, Variable name: index19,
  Default value: 1, Range:
  First entry for RA=19 h
- **"INDEX20": Type: J**, Variable name: index20,
  Default value: 1, Range:
  First entry for RA=20 h
- **"INDEX21": Type: J**, Variable name: index21,
  Default value: 1, Range:
  First entry for RA=21 h
- **"INDEX22": Type: J**, Variable name: index22,
  Default value: 1, Range:
  First entry for RA=22 h
- **"INDEX23": Type: J**, Variable name: index23,
  Default value: 1, Range:
  First entry for RA=23 h

35.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"RA(2000)" : Type D** "DEGREE " (1)
  Variable name root: Ra2000
  RA (J2000)
- **"DEC(2000)" : Type D** "DEGREE " (1)
  Variable name root: Dec2000
  Dec (J2000)
- **"PEAK INT " : Type E** "JY/BEAM " (1)
  Variable name root: PeakInt
  Peak Ipol
- **"MAJOR AX" : Type E** "DEGREE " (1)
  Variable name root: MajorAxis
  Fitted major axis size
- **"MINOR AX" : Type E** "DEGREE " (1)
  Variable name root: MinorAxis
  Fitted minor axis siz
- **"POSANGLE" : Type E** "DEGREE " (1)
  Variable name root: PosAngle
  Fitted PA
- **"Q CENTER" : Type E** "JY/BEAM " (1)
  Variable name root: QCenter
  Center Q flux density
- **"U CENTER" : Type E** "JY/BEAM " (1)
  Variable name root: UCenter
  enter U flux density
- **"P FLUX" : Type E** "JY " (1)
  Variable name root: PFlux
  Integrated polarized flux density
- **"I RMS " : Type E** "JY/BEAM " (1)
  Variable name root: IRMS
  Ipol RMS uncertainty
- **"POL RMS " : Type E** "JY/BEAM " (1)
  Variable name root: PolRMS
  RMS (sigma) in Qpol and Upol

- **"RES RMS" : Type E** "JY/BEAM " (1)
  Variable name root: ResRMS
  RMS of Ipol residual
- **"RES PEAK" : Type E** "JY/BEAM " (1)
  Variable name root: ResPeak
  Peak in Ipol residual
- **"RES FLUX" : Type E** "JY " (1)
  Variable name root: ResFlux
  Integrated Ipol residual
- **"CENTER X" : Type E** "PIXEL " (1)
  Variable name root: CenterX
  Center x position in pixels in FIELD
- **"CENTER Y" : Type E** "PIXEL " (1)
  Variable name root: CenterY
  Center y position in pixels in FIELD
- **"FIELD " : Type A** " " (8)
  Variable name root: Field
  Name of survey field
- **"JD PROCESSED" : Type J** "DAYS " (1)
  Variable name root: JDProcess
  Julian date on which entry was derived from image.

35.1.5 Modification History

1. W. D. Cotton 02/03/2003
   Revision 1: Copied from AIPS

36 Class ObitTableVZ

ObitTableVZ Class

36.1 NVSS short format catalog file

Table name: VZ

36.1.1 Introduction

[This class contains tabular data and allows access. "AIPS VZ" contains a catalog of sources. An ObitTableVZ is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

36.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

36.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"REVISION": Type: J**, Variable name: revision,
  Default value: 1, Range:
  Revision number of the table definition
- **"REF_FREQ": Type: D**, Variable name: refFreq,
  Default value: 1.4e9, Range:
  The Reference frequency

36.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"RA(2000)" : Type D** "DEGREE " (1)
  Variable name root: Ra2000
  RA (J2000)
- **"DEC(2000)" : Type D** "DEGREE " (1)
  Variable name root: Dec2000
  Dec (J2000)
- **"PEAK INT " : Type E** "JY/BEAM " (1)
  Variable name root: PeakInt
  Peak Ipol
- **"QUAL " : Type J** " " (1)
  Variable name root: Quality
  Quality (crowding) measure, 0=¿ best, higher worse

36.1.5 Modification History

1. W. D. Cotton 02/03/2003
   Revision 1: Copied from AIPS

37 Class ObitTableWX

Weather table for UV data.

37.1 Weather table for UV data

Table name: WX

37.1.1 Introduction

[This class contains tabular data and allows access. "AIPS WX" tables contain weather information. An ObitTableWX is the front end to a persistent disk resident structure. Both FITS and AIPS cataloged data are supported. This class is derived from the ObitTable class.]

37.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

37.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV": Type: J**, Variable name: revision,
  Default value: 3, Range:
  Revision number of the table definition.
- **"OBSCODE": Type: A**, Variable name: obscode,
  Default value: "AA000", Range:
  Observation code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYYMMDD", Range:
  Reference date as "YYYYMMDD"

37.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable

name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time of the solution
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  Solution interval.
- **"ANTENNA NO." : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"SUBARRAY" : Type J** " " (1)
  Variable name root: SubA
  Subarray number.
- **"TEMPERATURE" : Type E** "CENTIGRA" (1)
  Variable name root: temperature
  Temperature
- **"PRESSURE" : Type E** "MILLIBAR" (1)
  Variable name root: pressure
  Pressure
- **"DEWPOINT" : Type E** "CENTIGRA" (1)
  Variable name root: dewpoint
  Dew Point
- **"WIND_VELOCITY" : Type E** "M/SEC " (1)
  Variable name root: windVelocity
  Wind velocity
- **"WIND_DIRECTION" : Type E** "DEGREES " (1)
  Variable name root: windDirection
  Wind direction (azimuth)
- **"WVR_H2O" : Type E** " " (1)
  Variable name root: wvrH2O
  Water vapor radiometer
- **"ONOS_ELECTRON" : Type E** " " (1)
  Variable name root: onosElectron
  Ionospheric electron something

37.1.5 Modification History

1. W. D. Cotton 22/02/2006
   Revision 1: Copied from AIPS

38 Class ObitTableIDI_ANTENNA

ObitTableAN Class

38.1 Antenna table for IDI uv data

Table name: IDI_ANTENNA

38.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_ANTENNA" contains information about the characteristics of antennas in a UV data set. Only FITS cataloged data are supported. Polarization calibration information is included.]

38.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

38.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"
- **"NOPCAL ": Type: J**, Variable name: numPCal,
  Default value: 4, Range: ()
  Number of polarization calibration constants
- **"POLTYPE ": Type: A**, Variable name: polType,
  Default value: " ", Range:
  Polarization parameterazation type, 'APPR', 'RAPPR', 'ORI-ELP'

38.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAY " (1)
  Variable name root: time
  Time.
- **"TIME_INTERVAL" : Type E** "DAY " (1)
  Variable name root: time_interval
  Time interval over which antenna characteristics valid.
- **"ANNAME " : Type A** " " (8)
  Variable name root: AntName
  Station name
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antenna_no
  Station number, used as an index in other tables, uv data

- **"ARRAY" : Type J** " " (1)
  Variable name root: array
  Subarray number
- **"FREQID" : Type J** " " (1)
  Variable name root: freqid
  Frequency group id
- **"NO_LEVELS" : Type J** " " (1)
  Variable name root: no_levels
  Quantization of data, no. levels.
- **"POLTYA " : Type A** " " (1)
  Variable name root: polTypeA
  Feed A feed poln. type 'R','L','X','Y', actually only one valid character.
- **"POLAA " : Type E** "DEGREES " (no_band)
  Variable name root: PolAngA
  Feed A feed position angle
- **"POLCALA" : Type E** " " (numPCal,no_band)
  Variable name root: PolCalA
  Feed A poln. cal parameter.
- **"POLTYB " : Type A** " " (1)
  Variable name root: polTypeB
  Feed B feed poln. type 'R','L','X','Y'
- **"POLAB " : Type E** "DEGREES " (no_band)
  Variable name root: PolAngB
  Feed B feed position angle
- **"POLCALB" : Type E** " " (numPCal,no_band)
  Variable name root: PolCalB
  Feed B poln. cal parameter
- **"BEAMFWHM" : Type E** "DEGREES " (no_band)
  Variable name root: BeamFWHM
  [OPTIONAL] Beam Full Width Half Maximum

38.1.5 Modification History

1. W. D. Cotton 28/04/2007
   Revision 1: Initial definition
2. W. D. Cotton 02/09/2009
   Revision 1: Update definition

39 Class ObitTableIDI_ARRAY_GEOMETRY

ObitTableIDI_ARRAY_GEOMETRY Class

39.1 IDI Array geometry table for uv data

Table name: IDI_ARRAY_GEOMETRY

39.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. IDI_ARRAY_GEOMETRY contains information about the locations of antennas in a UV data set. Also time information and the state of the Earth's orientation. Only FITS cataloged data are supported.

39.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

39.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"
- **"FRAME ": Type: A**, Variable name: frame,
  Default value: "GEOCENTRIC", Range:
  Coordinate system
- **"NUMORB ": Type: J**, Variable name: numOrb,
  Default value: 0, Range: ()
  Number of orbital parameters
- **"FREQ ": Type: D**, Variable name: Freq,
  Default value: 1.0, Range:
  Obs. Reference Frequency for subarray(Hz)
- **"TIMSYS": Type: A**, Variable name: TimeSys,
  Default value: "IAT", Range:
  Time system, 'IAT' or 'UTC'
- **"GSTIA0 ": Type: D**, Variable name: GSTiat0,
  Default value: 0.0, Range:
  GST at time=0 (degrees) on the reference date
- **"DEGPDY ": Type: D**, Variable name: DegDay,
  Default value: 360.0, Range:
  Earth rotation rate (deg/IAT day)
- **"UT1UTC ": Type: E**, Variable name: ut1Utc,
  Default value: , Range:
  UT1-UTC (time sec.)

- **"IATUTC ": Type: E**, Variable name: iatUtc,
  Default value: 0.0, Range:
  data time-UTC (time sec.)
- **"POLARX ": Type: E**, Variable name: PolarX,
  Default value: 0.0, Range:
  Polar position X (meters) on ref. date
- **"POLARY ": Type: E**, Variable name: PolarY,
  Default value: 0.0, Range:
  Polar position Y (meters) on ref. date
- **"ARRAYX ": Type: D**, Variable name: ArrayX,
  Default value: 0.0, Range:
  Array center X coord. (meters, earth center)
- **"ARRAYY ": Type: D**, Variable name: ArrayY,
  Default value: 0.0, Range:
  Array center Y coord. (meters, earth center)
- **"ARRAYZ ": Type: D**, Variable name: ArrayZ,
  Default value: 0.0, Range:
  Array center Z coord. (meters, earth center)

39.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"ANNAME " : Type A** " " (8)
  Variable name root: AntName
  Station name
- **"STABXYZ " : Type D** "METERS " (3)
  Variable name root: StaXYZ
  X,Y,Z offset from array center
- **"DERXYZ " : Type E** "METERS " (3)
  Variable name root: derXYZ
  look this up
- **"ORBPARM " : Type D** " " (numOrb)
  Variable name root: OrbParm
  Orbital parameters.
- **"NOSTA " : Type I** " " (1)
  Variable name root: noSta
  Station number, used as an index in other tables, uv data

- **"MNTSTA " : Type J** " " (1)
  Variable name root: mntSta
  Mount type, 0=altaz, 1=equatorial, 2=orbiting, 3=dipole
- **"STAXOF " : Type E** "METERS " (3)
  Variable name root: staXof
  Axis offset (in 3D???)
- **"DIAMETER" : Type E** "METERS " (1)
  Variable name root: diameter
  [OPTIONAL] Antenna diameter

39.1.5 Modification History

1. W. D. Cotton 28/04/2007
   Revision 1: Initial definition
2. W. D. Cotton 02/09/2009
   Revision 1: Update definition

40 Class ObitTableIDI_BANDPASS

40.1 IDI UV data System Temperature Table

Table name: IDI_BANDPASS

40.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_BANDPASS" contains bandpass calibration data. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

40.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

40.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"
- **"NO_ANT ": Type: J**, Variable name: numAnt,
  Default value: , Range: ()
  The maximum antenna number
- **"NO_BACH ": Type: J**, Variable name: numBach,
  Default value: , Range: ()
  The number of spectral channels tabulated
- **"STRT_CHN": Type: J**, Variable name: strtChn,
  Default value: , Range: ()
  Data channel number (1-rel) of first channel in the table

40.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval.

- **"SOURCE_ID" : Type J** " " (1)
  Variable name root: SourID
  Source Identifier number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"ARRAY" : Type J** " " (1)
  Variable name root: Array
  Array number.
- **"FREQID " : Type J** " " (1)
  Variable name root: fqid
  Frequency ID number for row
- **"BANDWIDTH" : Type E** "HZ " (no_band)
  Variable name root: bandwidth
  Channel bandwidth per band
- **"BAND_FREQ" : Type D** "HZ " (no_band)
  Variable name root: band_freq
  Frequency per band
- **"REFANT_#NO_POL"" : Type J** " " (no_band)
  Variable name root: refant
  Frequency per band
- **"BREAL_#NO_POL" : Type E** " " (numBach,no_band)
  Variable name root: breal
  Real correction for poln # NO_POL
- **"BIMAG_#NO_POL" : Type E** " " (numBach,no_band)
  Variable name root: bimag
  Imaginary correction for poln # NO_POL

40.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

41 Class ObitTableIDI_CALIBRATION

41.1 IDI UV data Calibration Table

Table name: IDI_CALIBRATION

41.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_CALIBRATION" contains calibration correction for uv data. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

41.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

41.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"
- **"NO_ANT ": Type: J**, Variable name: numAnt,
  Default value: , Range: ()
  The maximum antenna number

41.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval.
- **"SOURCE_ID" : Type J** " " (1)
  Variable name root: SourID
  Source Identifier number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antNo
  Antenna number

- **"ARRAY" : Type J** " " (1)
  Variable name root: Array
  Array number.
- **"FREQID " : Type J** " " (1)
  Variable name root: fqid
  Frequency ID number for row, this is a random parameter in the uv data
- **"TSYS_#NO_POL" : Type E** "K " (no_band)
  Variable name root: TSys
  System temperature for poln # NO_POL
- **"TANT_#NO_POL" : Type E** "K " (no_band)
  Variable name root: TAnt
  Antenna temperature for poln # NO_POL
- **"SENSITIVITY_#NO_POL" : Type E** "K/JY "
  (no_band)
  Variable name root: sensitivity
  Sensitivity for poln # NO_POL
- **"PHASE_#NO_POL" : Type E** "RAD " (no_band)
  Variable name root: phase
  Phase for poln # NO_POL
- **"RATE_#NO_POL" : Type E** "SEC/SEC " (no_band)
  Variable name root: rate
  Rate of change of delay for poln # NO_POL
- **"DELAY_#NO_POL" : Type E** "SEC " (no_band)
  Variable name root: delay
  Delay for poln # NO_POL
- **"REAL_#NO_POL" : Type E** " " (no_band)
  Variable name root: real
  Real part of gain for poln # NO_POL
- **"IMAG_#NO_POL" : Type E** " " (no_band)
  Variable name root: imag
  Imaginary part of gain for poln # NO_POL
- **"WEIGHT_#NO_POL" : Type E** " " (no_band)
  Variable name root: weight
  Weight for poln # NO_POL
- **"REFANT_#NO_POL" : Type J** " " (no_band)
  Variable name root: refant
  Reference antenna for poln # NO_POL

41.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

42 Class ObitTableIDI_FLAG

42.1 IDI UV data Flag Table

Table name: IDI_FLAG

42.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_FLAG" contains descriptions of data to be ignored in uv data. An ObitTableIDI_FLAG is the front end to a persistent disk resident structure. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

42.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

42.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"

42.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"SOURCE_ID" : Type J** " " (1)
  Variable name root: SourID
  Source Identifier number
- **"ARRAY" : Type J** " " (1)
  Variable name root: Array
  Array number.
- **"ANTS" : Type J** " " (2)
  Variable name root: ants
  Antenna numbers of baseline, 0 means any
- **"FREQID " : Type J** " " (1)
  Variable name root: fqid
  Frequency ID number for row, this is a random parameter in the uv data
- **"TIMERANG" : Type E** "DAY " (2)
  Variable name root: timerange
  Beginning and end times

- **"BANDS" : Type J** " " (no_band)
  Variable name root: bands
  Band flag, not 0 means corresponding band flagged
- **"CHANS" : Type J** " " (2)
  Variable name root: chans
  Lowest and highest (1-rel) spectral channel numbers
- **"PFLAGS" : Type J** " " (4)
  Variable name root: pflags
  Polarization flags, non 0 means corresponding Stokes is flagged
- **"REASON" : Type A** " " (40)
  Variable name root: reason
  Reason for flagging
- **"SEVERITY" : Type J** " " (4)
  Variable name root: severity
  Severity, -1:no level, 0:known bad, 1:probably bad, 2:may be bad

42.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

43 Class ObitTableIDI_FREQUENCY

43.1 IDI UV data Frequency Table

Table name: IDI_FREQUENCY

43.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_FREQUENCY" contains frequency related information for "Bands" in uv data. A "Band" is a construct that allows sets of arbitrarily spaced frequencies. An ObitTableIDI_FREQUENCY is the front end to a persistent disk resident structure. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

43.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

43.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"

43.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"FREQID " : Type J** " " (1)
  Variable name root: fqid
  Frequency ID number for row, this is a random parameter in the uv data
- **"BANDFREQ" : Type D** "HZ " (no_band)
  Variable name root: bandfreq
  Offset from reference frequency for each Band
- **"CH_WIDTH" : Type E** "HZ " (no_band)
  Variable name root: chWidth
  Bandwidth of an individual channel, now always written and read as a signed value
- **"TOTAL_BANDWIDTH " : Type E** "HZ " (no_band)
  Variable name root: totBW
  Total bandwidth of the IF, now written and read as an unsigned value
- **"SIDEBAND " : Type J** " " (no_band)
  Variable name root: sideBand
  Sideband of the IF (-1 => lower, +1 => upper), now always written and read as +1

43.1.5 Modification History

1. W. D. Cotton 28/04/2007
   Revision 1: Initial definition

2. W. D. Cotton 02/09/2009
   Revision 1: Update definition

44 Class ObitTableIDI_GAIN_CURVE

44.1 IDI UV data Gain Curve Table

Table name: IDI_GAIN_CURVE

44.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_GAIN_CURVE" contains antenna gain as a function of pointing position. An ObitTableIDI_GAIN_CURVE is the front end to a persistent disk resident structure. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

44.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

44.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands

- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels
- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"
- **"NO_TABS": Type: J**, Variable name: numTabs,
  Default value: , Range: ()
  The number of tabulated terms

44.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X**=bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"ARRAY" : Type J** " " (1)
  Variable name root: Array
  Array number.
- **"FREQID " : Type J** " " (1)
  Variable name root: fqid
  Frequency ID number for row, this is a random parameter in the uv data

- **"TYPE_#NO_POL" : Type J** " " (no_band)
  Variable name root: type
  Gain curve type for poln # NO_POL
- **"NTERM_#NO_POL" : Type J** " " (no_band)
  Variable name root: nterm
  Number of terms for poln # NO_POL
- **"X_TYP_#NO_POL" : Type J** " " (no_band)
  Variable name root: x_typ
  X value types for poln # NO_POL
- **"Y_TYP_#NO_POL" : Type J** " " (no_band)
  Variable name root: y_typ
  Y value types for poln # NO_POL
- **"X_VAL_#NO_POL" : Type E** " " (no_band)
  Variable name root: x_val
  X values for poln # NO_POL
- **"Y_VAL_#NO_POL" : Type E** " " (numTabs,no_band)
  Variable name root: y_val
  Y values for poln # NO_POL
- **"GAIN_#NO_POL" : Type E** " " (numTabs,no_band)
  Variable name root: gain
  for poln # NO_POL
- **"SENS_#NO_POL" : Type E** " " (no_band)
  Variable name root: sens
  for poln # NO_POL

44.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

45 Class ObitTableIDI_INTERFEROMETER_MODEL

45.1 IDI UV data Interferometer Model Table

Table name: IDI_INTERFEROMETER_MODEL

45.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_INTERFEROMETER_MODEL" contains the correlator model used for the data. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

45.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

45.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations
- **"NPOLY ": Type: J**, Variable name: npoly,
  Default value: 1, Range: ()
  Number of polynomial parameters
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands

- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels
- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"

45.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"TIME_INTERVAL " : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval of record
- **"SOURCE_ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number

- **"ARRAY " : Type J** " " (1)
  Variable name root: Array
  Array number
- **"FREQID" : Type J** " " (1)
  Variable name root: FreqID
  Frequency ID
- **"I.FAR.ROT" : Type E** "RAD/M**2" (1)
  Variable name root: IFR
  Ionospheric Faraday Rotation
- **"FREQ.VAR" : Type E** "HZ " (no_band)
  Variable name root: FreqVar
  FREQ.VAR (?)
- **"PDELAY_#NO_POL " : Type D** "SECONDS " (npoly,no_band)
  Variable name root: PDelay
  Phase delay
- **"GDELAY_#NO_POL " : Type D** "SECONDS " (npoly,no_band)
  Variable name root: GDelay
  Group delay
- **"PRATE_#NO_POL " : Type D** "SECONDS " (npoly,no_band)
  Variable name root: PRate
  Phase delay rate
- **"GRATE_#NO_POL " : Type D** "SECONDS " (npoly,no_band)
  Variable name root: GRate
  Group delay rate
- **"DISP_#NO_POL " : Type E** "SECONDS " ()
  Variable name root: Disp
  Dispersion
- **"DDISP_#NO_POL " : Type E** "SEC/SEC " ()
  Variable name root: DRate
  Dispersion rate

45.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

46 Class ObitTableIDI_PHASE_CAL

46.1 IDI UV data Interferometer Model Table

Table name: IDI_PHASE-CAL

46.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_PHASE_CAL" contains instrumental phase calibration data Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

46.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

46.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"
- **"NO_TABS ": Type: J**, Variable name: numTones,
  Default value: 1, Range: ()
  Number of tones used

46.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"TIME_INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  The spanned time.
- **"SOURCE_ID " : Type J** " " (1)
  Variable name root: SourID
  Source ID number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antennaNo
  Antenna number

- **"ARRAY " : Type J** " " (1)
  Variable name root: Array
  Array number
- **"FREQID" : Type J** " " (1)
  Variable name root: FreqID
  Frequency ID
- **"CABLE_CAL" : Type D** "SECONDS " (1)
  Variable name root: CableCal
  CABLE_CAL
- **"STATE_#NO_POL " : Type E** "PERCENT " (4,no_band)
  Variable name root: State
  State counts(?)
- **"PC_FREQ_#NO_POL " : Type D** "HZ " (numTones,no_band)
  Variable name root: PCFreq
  Frequencies of the phase tones
- **"PC_REAL_#NO_POL " : Type E** " " (numTones,no_band)
  Variable name root: PCReal
  Real part of tone phase
- **"PC_IMAG_#NO_POL " : Type E** " " (numTones,no_band)
  Variable name root: PCImag
  Imaginary part of tone phase
- **"PC_RATE_#NO_POL " : Type E** "SEC/SEC " (numTones,no_band)
  Variable name root: PCRate
  Tone rate phase

46.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

47 Class ObitTableIDI_SOURCE

ObitTableIDI_SOURCE Class

47.1 IDI Source table for UV data

Table name: IDI_SOURCE

47.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_SOURCE" contains information about astronomical sources. An ObitTableIDI_SOURCE is the front end to a persistent disk resident structure. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

47.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

47.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"

47.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"SOURCE_ID" : Type J** " " (1)
  Variable name root: SourID
  Source ID
- **"SOURCE " : Type A** " " (16)
  Variable name root: Source
  Source name
- **"QUAL " : Type J** " " (1)
  Variable name root: Qual
  Source qualifier
- **"CALCODE " : Type A** " " (4)
  Variable name root: CalCode
  Calibrator code
- **"FREQID " : Type J** " " (1)
  Variable name root: FreqID
  Frequency group ID
- **"IFLUX " : Type E** "JY " (no_band)
  Variable name root: IFlux
  Total Stokes I flux density per band

- **"QFLUX " : Type E** "JY " (no_band)
  Variable name root: QFlux
  Total Stokes Q flux density per band
- **"UFLUX " : Type E** "JY " (no_band)
  Variable name root: UFlux
  Total Stokes U flux density per band
- **"VFLUX " : Type E** "JY " (no_band)
  Variable name root: VFlux
  Total Stokes V flux density per band
- **"ALPHA " : Type E** " " (no_band)
  Variable name root: alpha
  Spectral index per IF
- **"FREQOFF " : Type E** "HZ " (no_band)
  Variable name root: FreqOff
  Frequency offset (Hz) from band nominal per band
- **"RAEPO " : Type D** "DEGREES " (1)
  Variable name root: RAMean
  Right ascension at mean EPOCH (actually equinox)
- **"DECEPO " : Type D** "DEGREES " (1)
  Variable name root: DecMean
  Declination at mean EPOCH (actually equinox)
- **"EQUINOX " : Type A** " " (8)
  Variable name root: Equinox
  Mean Epoch (really equinox) for position in yr. since year 0.0
- **"RAAPP " : Type D** "DEGREES " (1)
  Variable name root: RAApp
  Apparent Right ascension
- **"DECAPP " : Type D** "DEGREES " (1)
  Variable name root: DecApp
  Apparent Declination
- **"SYSVEL " : Type D** "M/SEC " (no_band)
  Variable name root: SysVel
  Systemic velocity per Band (IF)
- **"VELTYP " : Type A** " " (8)
  Variable name root: VelTyp
  Velocity type
- **"VELDEF " : Type A** " " (8)
  Variable name root: VelDef
  Velocity definition 'RADIO' or 'OPTICAL'

- **"RESTFREQ" : Type D** "HZ " (no_band)
  Variable name root: RestFreq
  Line rest frequency per Band (IF)
- **"PMRA " : Type D** "DEG/DAY " (1)
  Variable name root: PMRa
  Proper motion (deg/day) in RA
- **"PMDEC " : Type D** "DEG/DAY " (1)
  Variable name root: PMDec
  Proper motion (deg/day) in declination
- **"PARALLAX" : Type E** "ARCSEC " (1)
  Variable name root: parallax
  Parallax
- **"EPOCH" : Type D** "YEAR " (1)
  Variable name root: Epoch
  [OPTIONAL]Epoch of position

47.1.5 Modification History

1. W. D. Cotton 28/04/2007
   Revision 1: Initial definition
2. W. D. Cotton 02/09/2009
   Revision 1: Update definition

48 Class ObitTableIDI_SYSTEM_TEMPERATURE

48.1 IDI UV data System Temperature Table

Table name: IDI_SYSTEM_TEMPERATURE

48.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_SYSTEM_TEMPERATURE" contains measured system and/or antenna temperatures Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

48.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

48.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_POL ": Type: J**, Variable name: numPol,
  Default value: , Range: (1,2)
  The number of polarizations
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"

48.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval.
- **"SOURCE_ID" : Type J** " " (1)
  Variable name root: SourID
  Source Identifier number
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"ARRAY" : Type J** " " (1)
  Variable name root: Array
  Array number.

- **"FREQID " : Type J** " " (1)
  Variable name root: fqid
  Frequency ID number for row, this is a random parameter in the uv data
- **"TSYS_#NO_POL" : Type E** "K " (no_band)
  Variable name root: TSys
  System temperature for poln # NO_POL
- **"TANT_#NO_POL" : Type E** "K " (no_band)
  Variable name root: TAnt
  Antenna temperature for poln # NO_POL

48.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

49 Class ObitTableIDI_UV_DATA

ObitTableIDI_UV_DATA Class

49.1 IDI UV data

Table name: IDI_UV_DATA

49.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. An IDI_UV_DATA table radio interferometer observational data.]

49.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

49.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 1, Range:
  Table format revision number
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands (IF)
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"
- **"NMATRIX ": Type: J**, Variable name: nmatrix,
  Default value: 1, Range:
  Number of UV data matricies
- **"MAXIS ": Type: J**, Variable name: maxis,
  Default value: 1, Range:
  Number of UV data matrix axes
- **"EQUINOX ": Type: A**, Variable name: Equinox,
  Default value: "J2000 ", Range:
  [OPTIONAL] System and mean equinox of coordinates
- **"WEIGHTYP ": Type: A**, Variable name: WeighTyp,
  Default value: "NORMAL", Range:
  [OPTIONAL] Type of data weights
- **"TMATX01": Type: L**, Variable name: tmatx01,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 1 has the visibility matrix
- **"TMATX02": Type: L**, Variable name: tmatx02,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 2 has the visibility matrix
- **"TMATX03": Type: L**, Variable name: tmatx03,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 3 has the visibility matrix
- **"TMATX04": Type: L**, Variable name: tmatx04,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 4 has the visibility matrix

- **"TMATX05": Type: L**, Variable name: tmatx05,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 5 has the visibility matrix
- **"TMATX06": Type: L**, Variable name: tmatx06,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 6 has the visibility matrix
- **"TMATX07": Type: L**, Variable name: tmatx07,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 7 has the visibility matrix
- **"TMATX08": Type: L**, Variable name: tmatx08,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 8 has the visibility matrix
- **"TMATX09": Type: L**, Variable name: tmatx09,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 9 has the visibility matrix
- **"TMATX10": Type: L**, Variable name: tmatx10,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 10 has the visibility matrix
- **"TMATX11": Type: L**, Variable name: tmatx11,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 11 has the visibility matrix
- **"TMATX12": Type: L**, Variable name: tmatx12,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 12 has the visibility matrix
- **"TMATX13": Type: L**, Variable name: tmatx13,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 13 has the visibility matrix
- **"TMATX14": Type: L**, Variable name: tmatx14,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 14 has the visibility matrix
- **"TMATX15": Type: L**, Variable name: tmatx15,
  Default value: FALSE, Range:
  [OPTIONAL] If TRUE col. 15 has the visibility matrix
- **"MAXIS1": Type: J**, Variable name: maxis1,
  Default value: 1, Range: ()
  [OPTIONAL] Number of pixels on axis 1
- **"CTYPE1": Type: A**, Variable name: ctype1,
  Default value: " ", Range:
  [OPTIONAL] Type of data axis 1

- **"CDELT1": Type: D**, Variable name: cdelt1,
  Default value: 0.0, Range:
  [OPTIONAL] Increment in coordinate between pixels on axis 1
- **"CRPIX1": Type: E**, Variable name: crpix1,
  Default value: 1.0, Range:
  [OPTIONAL] Reference pixel on axis 1
- **"CRVAL1": Type: D**, Variable name: crval1,
  Default value: 0.0, Range:
  [OPTIONAL] Reference coordinate on axis 1
- **"MAXIS2": Type: J**, Variable name: maxis2,
  Default value: 1, Range: ()
  [OPTIONAL] Number of pixels on axis 2
- **"CTYPE2": Type: A**, Variable name: ctype2,
  Default value: " ", Range:
  [OPTIONAL] Type of data axis 2
- **"CDELT2": Type: D**, Variable name: cdelt2,
  Default value: 0.0, Range:
  [OPTIONAL] Increment in coordinate between pixels on axis 2
- **"CRPIX2": Type: E**, Variable name: crpix2,
  Default value: 1.0, Range:
  [OPTIONAL] Reference pixel on axis 2
- **"CRVAL2": Type: D**, Variable name: crval2,
  Default value: 0.0, Range:
  [OPTIONAL] Reference coordinate on axis 2
- **"MAXIS3": Type: J**, Variable name: maxis3,
  Default value: 1, Range: ()
  [OPTIONAL] Number of pixels on axis 3
- **"CTYPE3": Type: A**, Variable name: ctype3,
  Default value: " ", Range:
  [OPTIONAL] Type of data axis 3
- **"CDELT3": Type: D**, Variable name: cdelt3,
  Default value: 0.0, Range:
  [OPTIONAL] Increment in coordinate between pixels on axis 3
- **"CRPIX3": Type: E**, Variable name: crpix3,
  Default value: 1.0, Range:
  [OPTIONAL] Reference pixel on axis 3
- **"CRVAL3": Type: D**, Variable name: crval3,
  Default value: 0.0, Range:
  [OPTIONAL] Reference coordinate on axis 3

- **"MAXIS4": Type: J**, Variable name: maxis4,
  Default value: 1, Range: ()
  [OPTIONAL] Number of pixels on axis 4
- **"CTYPE4": Type: A**, Variable name: ctype4,
  Default value: " ", Range:
  [OPTIONAL] Type of data axis 4
- **"CDELT4": Type: D**, Variable name: cdelt4,
  Default value: 0.0, Range:
  [OPTIONAL] Increment in coordinate between pixels on axis 4
- **"CRPIX4": Type: E**, Variable name: crpix4,
  Default value: 1.0, Range:
  [OPTIONAL] Reference pixel on axis 4
- **"CRVAL4": Type: D**, Variable name: crval4,
  Default value: 0.0, Range:
  [OPTIONAL] Reference coordinate on axis 4
- **"MAXIS5": Type: J**, Variable name: maxis5,
  Default value: 1, Range: ()
  [OPTIONAL] Number of pixels on axis 5
- **"CTYPE5": Type: A**, Variable name: ctype5,
  Default value: " ", Range:
  [OPTIONAL] Type of data axis 5
- **"CDELT5": Type: D**, Variable name: cdelt5,
  Default value: 0.0, Range:
  [OPTIONAL] Increment in coordinate between pixels on axis 5
- **"CRPIX5": Type: E**, Variable name: crpix5,
  Default value: 1.0, Range:
  [OPTIONAL] Reference pixel on axis 5
- **"CRVAL5": Type: D**, Variable name: crval5,
  Default value: 0.0, Range:
  [OPTIONAL] Reference coordinate on axis 5
- **"MAXIS6": Type: J**, Variable name: maxis6,
  Default value: 1, Range:
  [OPTIONAL] Number of pixels on axis 6
- **"CTYPE6": Type: A**, Variable name: ctype6,
  Default value: " ", Range:
  [OPTIONAL] Type of data axis 6
- **"CDELT6": Type: D**, Variable name: cdelt6,
  Default value: 0.0, Range:
  [OPTIONAL] Increment in coordinate between pixels on axis 6

- **"CRPIX6": Type: E**, Variable name: crpix6,
  Default value: 1.0, Range:
  [OPTIONAL] Reference pixel on axis 6
- **"CRVAL6": Type: D**, Variable name: crval6,
  Default value: 0.0, Range:
  [OPTIONAL] Reference coordinate on axis 6
- **"MAXIS7": Type: J**, Variable name: maxis7,
  Default value: 1, Range:
  [OPTIONAL] Number of pixels on axis 7
- **"CTYPE7": Type: A**, Variable name: ctype7,
  Default value: " ", Range:
  [OPTIONAL] Type of data axis 7
- **"CDELT7": Type: D**, Variable name: cdelt7,
  Default value: 0.0, Range:
  [OPTIONAL] Increment in coordinate between pixels on axis 7
- **"CRPIX7": Type: E**, Variable name: crpix7,
  Default value: 1.0, Range:
  [OPTIONAL] Reference pixel on axis 7
- **"CRVAL7": Type: D**, Variable name: crval7,
  Default value: 0.0, Range:
  [OPTIONAL] Reference coordinate on axis 7
- **"DATE-OBS": Type: A**, Variable name: dateObs,
  Default value: "0000-00-00", Range:
  [OPTIONAL] Observing date as YYYY-MM-DD
- **"TELESCOP": Type: A**, Variable name: teles,
  Default value: , Range:
  [OPTIONAL] Telescope used
- **"OBSERVER": Type: A**, Variable name: observer,
  Default value: , Range:
  [OPTIONAL] Observer making the observations
- **"VIS_SCAL": Type: E**, Variable name: visScale,
  Default value: 1.0, Range:
  [OPTIONAL] Scaling factor for visibilities
- **"SORT ": Type: A**, Variable name: sort,
  Default value: " ", Range:
  [OPTIONAL] Sort order of data

49.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X**=bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"UU " : Type E** "SECONDS " (1)
  Variable name root: uu
  U component of baseline.
- **"VV " : Type E** "SECONDS " (1)
  Variable name root: vv
  V component of baseline.
- **"WW " : Type E** "SECONDS " (1)
  Variable name root: ww
  W component of baseline.
- **"DATE " : Type D** "DAYS " (1)
  Variable name root: date
  Julian Date
- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time.
- **"BASELINE" : Type J** " " (1)
  Variable name root: Baseline
  Interferometer baseline as ant 1*256+ant1 (1-rel).
- **"ARRAY " : Type J** " " (1)
  Variable name root: Array
  Array number (1-rel)
- **"FILTER " : Type J** " " (1)
  Variable name root: Filter
  [OPTIONAL] VLBA Filter number.
- **"SOURCE_ID" : Type J** " " (1)
  Variable name root: Source
  [OPTIONAL] Source ID, index into SOURCE table.
- **"FREQID " : Type J** " " (1)
  Variable name root: FreqID
  [OPTIONAL] Frequency group ID.
- **"INTTIM" : Type E** "SECONDS " (1)
  Variable name root: IntTim
  [OPTIONAL] The integration time.

- **"WEIGHT" : Type E** " " (maxis2,no_band)
  Variable name root: Weight
  Weight per IF/poln.
- **"GATEID " : Type J** " " (1)
  Variable name root: GateID
  [OPTIONAL] VLBA Pulsar gate ID.
- **"FLUX " : Type E** " " (maxis1,maxis2,maxis3,maxis4,maxis5)
  Variable name root: Flux
  Visibility matrix

49.1.5 Modification History

1. W. D. Cotton 28/04/2007
   Revision 1: Initial definition
2. W. D. Cotton 02/09/2009
   Revision 1: Update definition

50 Class ObitTableIDI_WEATHER

50.1 IDI UV data weather Table

Table name: IDI_WEATHER

50.1.1 Introduction

[This class contains tabular data and allows access. This table is part of the IDI uv data format. "IDI_WEATHER" contains weather data. An ObitTableIDI_WEATHER is the front end to a persistent disk resident structure. Only FITS cataloged data are supported. This class is derived from the ObitTable class.]

50.1.2 Overview

In memory tables are stored in a fashion similar to how they are stored on disk - in large blocks in memory rather than structures. Due to the word alignment requirements of some machines, they are stored by order of the decreasing element size: double, float long, int, short, char rather than the logical order. The details of the storage in the buffer are kept in the ObitTableDesc.

50.1.3 Keywords

The following keywords must follow the required bintable keywords but the order is otherwise arbitrary. Allowed data types are: **I**=integer(16-bit), **J**=integer(32-bit), **E**=real, **D**=double, **A**=character, **L**=logical. "Variable name" is the name the keyword is given in software and "Range" if present indicates a structural keyword, "()" indicates all values are allowed and "(n,m)" indicates values from n to m. This latter is used to indicate sufixes for column lables.

- **"TABREV ": Type: J**, Variable name: tabrev,
  Default value: 2, Range:
  Table format revision number
- **"NO_STKD ": Type: J**, Variable name: no_stkd,
  Default value: 1, Range:
  Number of Stokes parameters
- **"STK_1 ": Type: J**, Variable name: stk_1,
  Default value: -1, Range:
  First Stokes parameter
- **"NO_BAND ": Type: J**, Variable name: no_band,
  Default value: 1, Range: ()
  Number of frequency bands
- **"NO_CHAN ": Type: J**, Variable name: no_chan,
  Default value: 1, Range:
  Number of frequency channels

- **"REF_FREQ": Type: D**, Variable name: ref_freq,
  Default value: 1.0, Range:
  Reference frequency (Hz)
- **"CHAN_BW ": Type: D**, Variable name: chan_bw,
  Default value: 1.0, Range:
  Channel bandwidth (Hz)
- **"REF_PIXL": Type: J**, Variable name: ref_pixl,
  Default value: 1.0, Range:
  Reference frequency bin
- **"OBSCODE ": Type: A**, Variable name: obscode,
  Default value: " ", Range:
  Observation project code
- **"ARRNAM ": Type: A**, Variable name: ArrName,
  Default value: , Range:
  Array name
- **"RDATE ": Type: A**, Variable name: RefDate,
  Default value: "YYYY-MM-DD", Range:
  Reference date as "YYYY-MM-DD"

50.1.4 Columns

The order of the columns is arbitrary. Allowed data types are: **I**=16 bit integer, **J**=32 bit integer, **E**=real, **D**=double, **C**=Complex (32 bit each), **M**=double complex (64 bit each), **A**=character, **L**=logical, **X=**bit array, **B**=byte array. Multidimensional arrays use the `TDIMnnn` keyword convention.
The first line of each entry gives the name:type, units and dimensionality of the entry. "Variable name" is the name the keyword is given in software, possibly with a suffix if the column label ends in #xxx where xxx is a keyword.

- **"TIME " : Type D** "DAYS " (1)
  Variable name root: Time
  The center time
- **"TIME INTERVAL" : Type E** "DAYS " (1)
  Variable name root: TimeI
  Time interval.
- **"ANTENNA_NO" : Type J** " " (1)
  Variable name root: antNo
  Antenna number
- **"TEMPERATURE" : Type E** "DEG C " (1)
  Variable name root: temperature
  Air temperature
- **"PRESSURE" : Type E** "MILLIBAR " (1)
  Variable name root: pressure
  Air pressure.

- **"DEWPOINT" : Type E** "DEG C " (1)
  Variable name root: dewpoint
  Dew point
- **"WIND_VELOCITY" : Type E** "M/S " (1)
  Variable name root: wind_velocity
  Wind velocity
- **"WIND_DIRECTION" : Type E** "DEC " (1)
  Variable name root: wind_direction
  Wind direction, east from north
- **"WVR_H2O" : Type E** "M**-2 " (1)
  Variable name root: wvr_h2o
  Water column density
- **"IONOS_ELECTRON" : Type E** "M**-2 " (1)
  Variable name root: ionos_electron
  Electron column density

50.1.5 Modification History

1. W. D. Cotton 02/09/2009
   Revision 1: Initial definition

References

Cotton, W. D., Tody, D., and Pence, W. D. 1995, A&AS, 113, 159–166.
Flatters, C., 1998, AIPS Memo No, 102, NRAO.
Wells, D. C., Greisen, E. W., and Harten, R. H. 1981, A&AS, 44, 363.