NOTE 260 – Table System File Formats

Ger van Diepen, ASTRON Dwingeloo

November 10, 2010

Abstract

The Casacore Table System (CacTS) uses several files to store data in. Each storage manager has its own files. The document describes the formats of all files that can be used by the Table system.

A pdf version of this note is available.

Contents

1 Introduction

The Casacore Table System (CacTS) is an RDBMS-like system to store data in tables consisting of a number of columns and rows. A table and each column can have an associated set of keywords to define global data (like subtables or the unit of a column containing numeric data). Also called TAOSS (Tabular Array-Oriented Storage System)

Besides the usual scalar data, a table column or keyword can hold N-dimensional arrays. The data type of a scalar or array can be Bool, Char, uChar, Short, uShort, Int, uInt, Float, Double, Complex, DComplex, or String. Furthermore a keyword can be an Int64,

The keywords are meant for small amounts of data. The bulk data will be stored in columns. There is a strict distinction between the logical and physical model for the data columns. The table description defines the logical model, while data managers (storage managers and virtual data managers) implement the physical model.

On disk a CaCTS table is a directory containing a number of files to hold data and meta data. The formats of these files are quite varying and can be quite complicated.
Furthermore a table can have a zero or more subtables that are usually stored as a subdirectory of the main table. A good example of this is the casacore MeasurementSet (see note 229).

A CaCTS table can be a reference to one or more other tables which is comparable to a view in a relational data base. CaCTS supports a few types of tables.

- A PlainTable is the basic table type. It contains all data.
- A RefTable is a referencing table. It only references rows in another table. Usually it is the result of a select, sort, or iteration of another table. The subtables of a RefTable are the same as those of its parent; it references them.
- A ConcatTable is the virtual concatenation of tables with an identical structure (identical columns). It only references the tables to be concatenated. The subtables of a ConcatTable can be concatenated at will. It means that a subtable can be the subtable of the first table or a concatenation of the subtable in all tables.

All table types consist of a directory with the name of the table and several files. The following two files are always present. A PlainTable can contain several more files.

- `table.dat` is a binary file holding the table definition. The first part of its contents is the same for all table types. The remainder is type specific.
- `table.info` is a text file containing a brief description of the table.

This document describes the format of all files supported natively by CaCTS. CaCTS can use arbitrary storage managers dynamically loaded from a shared library. The file formats of such third party storage managers are not described in this document.

CaCTS is designed such that it is fully backward compatible. Even tables from the very first days can be read back despite the fact that the format has changed considerably.
CaCTS understands both little and big endian data representations. At table creation time it is decided which one to use.

2 AipsIO

CacTS makes heavily use of the AipsIO class to store its meta data. Therefore a brief description of the AipsIO format is given.

AipsIO is basically a mechanism to write a C++ object into a stream of bytes and to read it back. The format consists of a header followed by the object's data. It can be nested arbitrarily deeply.
The header contains the following fields:

Data Type	Description
uInt	Object length
String	Object type
uInt	Version

The version can be used to make the software fully backward compatible in case an object changes over time. This feature is used a lot by CaCTS.
The object data are stored after the header as a stream of bytes. The data are unaligned in the stream, thus, say, an integer does not need to start on a multiple of 4 bytes.
The length of an object is the total length; it includes the length of the header and the length of possibly nested objects. Of course, each nested object has its own header.
The following data types are natively supported. Other data types (classes) can be supported by writing the appropriate shift functions (a la iostream).

Bool	1 byte
Char and uChar	1 byte
Short and uShort	2 bytes
Int and uInt	4 bytes
Int64 and uInt64	8 bytes
Float	4 bytes IEEE
Double	8 bytes IEEE
Complex	2 Floats
DComplex	2 Doubles
String	uInt length followed by its characters (can be length 0)
C-array	uInt length followed by its elements (can be length 0). A bool array is compressed to bits.

AipsIO can store the data in little or big endian order. In fact, it can also handle old VAX/VMS and IBM/360 data formats.
CaCTS only uses the canonical AipsIO format (which is big endian).

2.1 Array objects

Templated Array<T> objects are used quite heavily in casacore. In AipsIO they are stored as follows.

header	Array (version 3)
uInt	number of dimensions
ndim Int	shape
ndim Int	origin (only for version < 3)
T	all array values. Bools are compressed to bits.

- Casacore Array objects hold their data in Fortran order (fastest varying axis first). The shape and data are also stored that way.
- In principle the template parameter can be of any type. However, CaCTS only uses Arrays of the standard types.
- Until 1995 the Array class supported an origin. It has been removed because it was a source of errors. Theoretically an origin can still be present in Array objects in very old tables. If so, it is read and silently discarded.
- Before RTTI was fully supported in C++, casacore used its own RTTI functionality. At that time the name of the template parameter was made part of the AipsIO header, for example `Array<Int>`. After the standard C++ RTTI was used, this was not done anymore and an array of any type has the same header name `Array`. When reading an older Array object from AipsIO, the template part in the name is still recognized, but not used anymore.

2.2 IPosition object

An IPosition object defines the shape of an Array object. It will be stored in AipsIO as follows.

header	IPosition (version 1 or 2)
uInt	number of dimensions
ndim Int	shape (for version 1)
ndim Int64	shape (for version 2)

Until 2009 a shape element was represented as a 32-bit integer, thereafter as a 64-bit integer. To be as forward compatible as possible, an IPosition is still stored as version 1 if all elements in the shape have a value fitting in a 32-bit integer.

2.3 TableRecord object

A TableRecord object holds a set of keyword/value pairs. They are used in CaCTS to represent the keyword set attached to the table and each column.
A value can be a scalar or array of one the standard data types including complex and string values. Furthermore a value can be a TableRecord in itself; the nesting can be arbitrarily deep. Finally a value can be a Table object which is used to hold the subtables of a table.

A TableRecord is described by a RecordDesc object that also contains the descriptions of possible nested records. A TableRecord can have a fixed or variable format. In CaCTS only variable fomat TableRecords are used.

In the very early days of CaCTS a keyword set was stored in a very different way. Reading back such keyword sets is still supported in the code.

A TableRecord is stored in AipsIO as follows:

Data Type	Description
AipsIO header	TableRecord (version 1)
RecordDesc	The TableRecord description
Int	record type (CaCTS always uses variable format records)
any	All values are stored consecutevily. Their data types are defined in the RecordDesc. Scalars and arrays are stored in the standard AipsIO way. For a nested TableRecord only the values are stored, because its description is part of the record description. However, a nested empty TableRecord is stored as such. If a value is a Table, only the table name is stored. The name is tried to be made relative to the name of the parent table. In that way moving a table has no effect its subtable names. A relative name is prefixed with ./'/ if the full parent table name has to be used as directory, while the prefix ./ is used if the directory name of the parent table has to be used. Note that in all practical cases (like a MeasurementSet) ././ is used as prefix.

The RecordDesc is stored as follows:

Data Type	Description
AipsIO header	RecordDesc (version 2)
uInt	number of keywords
String	keyword name
Int	data type (from DataType.h)
Specific	specific info for a value containing an array, Table, or Record. - shape for an array (shape [-1] means it is not fixed). A shape is stored as a vector of Int values. - TableDesc name for a Table (empty name means it is not fixed). - description for a Record.
String	comment to explain the meaning of a keyword (not in version 1)

The latter 4 fields are repeated for each keyword.

3 PlainTable

A PlainTable object represent a basic CaCTS table with its own keywords, columns, and associated data managers. It is a directory containing several files named `table.*`.
Besides the files `table.dat` and `table.info` a PlainTable consists of the following files.

- `table.lock` is used to facilitate concurrent access to a table. Locks are acquired and released for this file. Furthermore it contains information telling another process if a table has changed.
- The files `table.f<i>*` are used by storage managers to store the data. The storage managers are numbered 0..n and use the number as the suffix `i` in the file name. Note that the numbers

do not need to be consecutive. This can happen in case a storage manager has been deleted or a virtual data manager is used.

3.1 General file table.dat

This file is the main table file. It contains the table definition, the keywords and their values, and the binding of columns to data managers. It is basically a single AipsIO object containing the following fields.

Data Type	Description
AipsIO header	Table (version 2)
uInt	number of table rows
uInt	endianness (0=little, 1=big)
String	table type (PlainTable)
TableDesc	table description and table keyword set
TableRecord	table keyword set (in version 1 only)
ColumnSet	column data manager info and column keyword sets

Note that the set of table and column keywords is part of the table description. Only in the very first CaCTS version (before 1995), the table keyword set was stored separately. For backward compatibility it is still possible to read such a keyword set, but it is not stored anymore.

It is instructive to use the UNIX `strings` command on a table.dat file. It shows all the strings mentioned in this section.

3.1.1 Table Description

The TableDesc object contains the table description defining the logical view of a table. It contains the name, data type, and some extra info of each column. Furthermore it contains the keyword set attached to the table and each column. Finally it contains a private keyword set that is used by the TableDesc object to hold some extra meta data.

Data Type	Description
AipsIO header	TableDesc (version 2)
String	name of table description
String	version info of table description
String	comment about table description
TableRecord	table keyword set
TableRecord	private table keyword set (not in version 1)
uInt	number of columns
ColumnDesc	info of each column

The info of each column is stored in a standard way.

Data Type	Description
uInt	ColumnDesc version (=1)
String	Type of column using the name of the object containing the description. It can have one of the following values: - ScalarColumnDesc<T - ArrayColumnDesc<T - ScalarRecordColumnDesc - SubTableDesc where T is a string defining the type. It can be: Bool, uChar, Short, uShort, Int, uInt, float, double, Complex, DComplex, and String.
uInt	BaseColumnDesc version (=1)
String	column name
String	comment
String	default data manager type
String	default data manager group
Int	data type (like TpInt from DataType.h¿. It must match the type in the column type string.
Int	column options 1 = direct array; meant for small fixed sized arrays for which a storage manager can decide to put it directly. 2 = undefined value can exist (not used). 4 = array has a fixed shape.
uInt	array dimensionality (-1 means scalar)
IPosition	fixed array shape (only for columns containing arrays)
uInt	maximum length of a string value (0 = no maximum)
TableRecord	column keyword set
	For a column containing scalars
uInt	ScalarColumnDesc version (=1)
T	default value (stored according to the data type)
	For a column containing arrays
uInt	ArrayColumnDesc version (=1)
Bool	dummy flag (always False); is present for backward compatibility
	For a column containing records
uInt	RecordScalarColumnDesc version (=1)
	For a column containing subtables
uInt	SubTableDesc version (=1)
String	TableDesc name
Bool	Flag telling how the description of the subtable is found. True = table description is by name, thus in a file with that name. False = table description is in the next field.
TableDesc	the description of the subtable (only if flag=False)

- A column containing records makes it possible to store arbitrary data in a column. It is handed as an AipsIO byte stream (vector of uChar) to the data managers.

- A column containing subtables is supported in the description, but it has never been fully implemented in CaCTS, so it cannot be used. This feature is only used in a test program.

3.1.2 ColumnSet

3.2 Info file table.info

This file is a text file giving some brief info about the table. It is read and written by class `TableInfo`.
It contains the following lines:

- The table type like `Type = MeasurementSet`
- The table subtype like`SubType =`
- Zero or more lines giving some more explanation.

3.3 Lock file table.lock

CaCTS supports concurrent access to a table. One writer or multiple readers can be active at the same time. A lock needs to be acquired to access the table, although there is a mode to bypass locking when reading. See note 256 for more info about how locking is done.

The first few bytes in the lock file are used by the concurrency mechanism of CaCTS to manage a read or write lock (on the entire table).

- The first byte is used to acquire/release a lock using the system's file locking mechanism (`fcntl`). The lock can be shared (for read) or exclusive (for write).
- The second byte is used to tell other processes that the table is opened. This is done by creating a shared lock on it.
- The third byte is used to tell other processes that a process holds a permanent lock on the table. It is also using a shared lock.
- The fourth till sixth byte are used for special internal lock purposes.

The lock file contains two data structures to assist processes in acquiring locks and synchronizing their internal data structures with table data changes made in another process. All data are written in big-endian format.

- A list of process identifications is kept in the first 260 bytes of the file. The list contains the host and process ids of processes wanting to acquire a lock, but cannot because another process is holding the lock.
  The list consists of 4-byte integers, where the first value contains the number of processes in the list. The list can contain up to 32 processes.
- Info telling if and how table data have changed. It is written when a write lock is released. It is used by a process acquiring a lock to synchronize its internal data objects.
  The info is written as an AipsIO stream described below. Bytes 260-263 of the lock file contain the length of the info stream. Thereafter the AipsIO stream is written.

Data Type	Description
AipsIO header	sync (version 1)
uInt	number of table rows
uInt	number of table columns
uInt	modify counter
uInt	table change counter. The counter is incremented if the table.dat file has changed (e.g. by adding a keyword).
Block<uint>	change counter per data manager. A counter is incremented if that data manager has written data.

A process acquiring a lock detects if the main table file or a data manager file has changed by comparing its change counters with the ones in the lock file.

3.4 Storage Managers

3.4.1 StandardStMan

- header block - index - data

3.4.2 IncrementalStMan

3.4.3 TiledStMan

- index - cell, column, shape stman

3.4.4 AipsIOStMan

3.5 Virtual Column Engines

4 RefTable

5 ConcatTable