Astronomical Data Analysis Software and Systems IV
ASP Conference Series, Vol. 77, 1995
R. A. Shaw, H. E. Payne, and J. J. E. Hayes, eds.

A Generic Data Exchange Scheme Between FITS Format and C Structures

W. Peng and T. Nicinski

Fermi National Accelerator Laboratory, PO Box 500,Batavia, IL 60510

Abstract. A flexible and efficient scheme allowing arbitrary FITS Binary and ASCII Tables to be converted to arbitrary C structures at run-time is presented. This scheme has been successfully implemented and used with Shiva[1] (Survey Human Interface and Visualization Environment), a package developed by Fermilab for the analysis of Sloan Digital Sky Survey data.

1. Introduction

The Sloan Digital Sky Survey (SDSS), for which Fermilab has been actively developing software and hardware, uses the Flexible Image Transport System (FITS) (NOST 1993) as the standard exchange format for survey data. Portions of the data are presented in FITS Binary and ASCII Tables. Accessing such arbitrary data from C structures, without knowing the FITS Table layout, can be difficult.

We have developed a versatile scheme that allows data transfer between FITS Tables and C data structures. This generic scheme uses two supporting structures: a TBLCOL to contain an arbitrary FITS Binary or ASCII Table (or both), and a translation table that maps TBLCOL to a user-specified C structure. FITS Tables are read into a TBLCOL structure. With a translation table filled in at run-time, C structures can be filled with data from TBLCOLs, and vice versa. This functionality is incorporated into Shiva, a package developed at Fermilab for analyzing SDSS data. The reading (writing) of arbitrary FITS Tables into (from) TBLCOLs and the translation of TBLCOL data to C structures are performed from the Shiva command line at run-time, without any compile-time knowledge of the FITS Tables and the C structures.

All primitive C data types, including characters, integers, floating point numbers, and strings, as well as arrays and structures of these types, are supported. Indirect data can also be accessed (through pointers).

---

[1] http://www-sdss.fnal.gov:8000/shiva/doc/www/shiva.home.html

2. TBLCOL Format[2]

Under Shiva, FITS Binary and ASCII Tables are read into and written from TBLCOLs. The TBLCOL format is flexible enough to accommodate any tabular data. Once data is in a TBLCOL, the originating FITS Table type is irrelevant. This makes it possible to read in a FITS ASCII Table and then write it out as a Binary Table, and vice versa (as long as the resulting FITS Table is legal).

The TBLCOL format uses three major structures to achieve its goal of supporting arbitrary tables: TBLCOL, ARRAY, and TBLFLD.

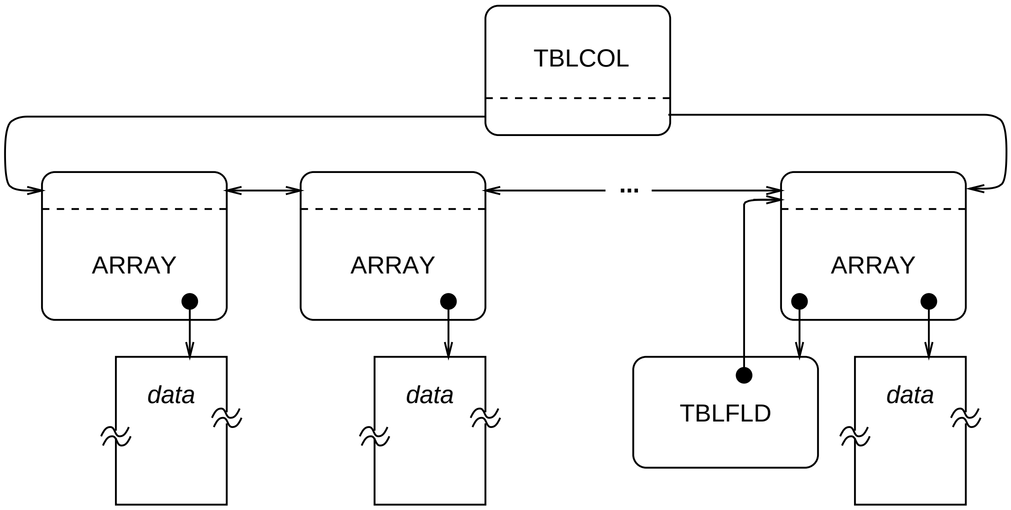

Figure 1. TBLCOL Format Components.

As a FITS Table is read in, each field is placed into an ARRAY. The TBLCOL structure simply heads the list of ARRAYs. An ARRAY points off to the FITS Table data, where each ARRAY element corresponds to the field data from a FITS Table row. The TBLFLD structure is optional, containing information about a field such as its name (akin to the FITS TTYPEn keyword), scaling and zeroing (FITS TSCALn and TZEROn keywords), etc. This organization allows quick and easy retrieval of data in a column/field oriented way. It also allows a FITS Table to be read into memory without any *a priori* knowledge of the FITS file contents or Table structure.

The ARRAY structure supports FITS Binary Tables having fields that are multidimensional arrays in themselves. The data type is *not* restricted only to primitives. Structures, and arrays of structures, can also be stored in the ARRAY and accessed properly. However, such use is not recommended if the TBLCOL is intended to be written out as a FITS Binary or ASCII Table (FITS does not permit such structures).

---

[2] http://www-sdss.fnal.gov:8000/shiva/doc/www/shTblHome.html

3. Translation Table[3]

TBLCOLs allow users to read in arbitrary FITS Binary or ASCII Tables. But, access to TBLCOL data is only efficient if it is processed on a field by field basis (it is relatively expensive to "bounce" to another field). The use of translation tables to move some or all data from a TBLCOL to a C structure can be used to circumvent this inefficiency. Users build translation tables at run-time, instructing how TBLCOL fields and C structure members are related. A translation table is a collection of textual entries of the form:

EntryType FldName C_MemName C_MemType OptInfo

where *EntryType* can be either "name" or "cont" and *OptInfo* contains optional dimension information. Each entry represents a mapping that associates a TBLCOL field, *FldName*, to a C structure member, *C_MemName*.

Data copying routines use this mapping, along with a C structure's *schema*[4] to properly copy between the TBLCOL and C structure (or vice versa). Type casting, checking structure member and TBLCOL field sizes, allocating memory, and traversing pointers are done transparently during the copy.

3.1. Primitive Data Types and Fixed-size Arrays

For primitive data types (such as characters, integers, and floating point numbers), the relation is a straightforward one-to-one mapping. The translation routines simply copy the data directly between a TBLCOL field and a C structure (with any appropriate type conversions). For example,

```
name RA_IN_DEG  ra  double
```

indicates that data from the TBLCOL field `RA_IN_DEG` be copied as a `double` precision floating point number to the `ra` member in a C structure, or vice versa.

Fixed-size arrays of primitive data types are handled in a similar fashion. Their size is already embedded in the C structure declaration and reflected in the structure's schema (see Section 4).

3.2. Dynamically Allocated Arrays

Non-trivial C structures can have, for example, arrays of C primitive types whose memory is allocated at run-time. When transferring data from TBLCOL, memory must be allocated properly for the receiving C structure. The size of this transfer is obtained from additional information in a translation table entry. For instance, a size of "5x10" indicates that the C structure member is a 2-dimensional array (5 by 10). When transferring data to TBLCOL, the TBLCOL field should have the appropriate space.

[3] http://www-sdss.fnal.gov:8000/shiva/doc/www/shSchema.html

[4] A *schema* describes a C structure at run-time. Applications can understand a C structure *without* needing to be compiled with the structure declaration. See Section 4 for more information.

3.3. Indirect Data

In practice, C structures have pointers to different memory areas. FITS does not support pointer data types in Tables. The translation tables take this into account through multiline entries. (a main name line followed by one or more continuation lines). Each line can have independent dimension information, imitating the process of traversing memory links to the ultimate data.

For example, consider the following two C structures

```
typedef struct {              typedef struct {
     ⋮                              ⋮
   REGION *reg;                   char *name;
     ⋮                              ⋮
} MY_STRUCT;                  } REGION;
```

The translation to match a FITS Table field, REG_NAME, could be

```
name  REG_NAME  reg   struct
cont  reg       name  string  -dimen=10
```

which states that, reg in MY_STRUCT is a pointer to a REGION object. A mapping between REG_NAME and reg→name is established. When transferring data to TBLCOL, the data pointed to by reg→name are copied. Likewise, when transferring from TBLCOL, two memory allocations are done (one each for reg and name) to ensure problem-free copying from REG_NAME to reg→name.

4. Schemas and Concluding Remarks

A *schema* is a run-time description of a C structure. It permits applications to understand a C structure *without* having been compiled with the structure declaration. During compilation, the Shiva environment parses C header files to generate schema for structures. Information about a structure, such as the member names, their sizes, offsets, dimensions, etc., are retained and are available at run-time. Currently, there are about 50 C structures used in Shiva. With the translation tables, passing data between arbitrary FITS Tables and these structures is possible. Applications built on top of Shiva also enjoy this capability.

Acknowledgments. This research is sponsored by DOE Contract number DE-AC02-76CHO3000.

References

NASA Office of Standards and Technology 1993, Definition of the Flexible Image Transport System (FITS) (Greenbelt, NASA/OSSA)