Astronomical Data Analysis Software and Systems IX
ASP Conference Series, Vol. 216, 2000
N. Manset, C. Veillet, and D. Crabtree, eds.

Data Access Layer to Support Variable Inputs Using C

M. Cresitello-Dittmar

Harvard-Smithsonian Center for Astrophysics, Cambridge, MA 02138

Abstract. This paper provides an overview of a software library developed at the Chandra X-Ray Observatory Science Center (CXC) to accommodate variability in the input data products to the aspect determination system. The issue arose that, depending on the decommutation templates used, the inputs to these tools would vary. This variation could be in the number of files, the columns contained in each file, and the form (type) of the column. We needed a layer of software that would dissociate a data column (element) from any particular file. It would also need to handle timing variations for inputs provided at different sample rates.

The software described in this paper is a library of functions and structures written in C, that define and manipulate data elements. Tools using this library provide a set of input files containing the required elements. The library functions sort the input files by type, locate the required elements in those files and provide access to them in a time-correlated fashion. The design of the system also makes it easy to define and create new data products.

1. Introduction

This data access layer is built on the concept of a data "element". A data element is a completely self-contained definition of a FITS table column, image, or keyword. A series of nested structures combine to completely identify each data element. These structures define things like the file and extension the element is in, the column number (if applicable), it's type and dimensionality. By linking together a series of data elements, a data product can be constructed, or conversely, a set of data products can be combined to function as single series of elements.

Since each element carries a complete definition, they can be manipulated independently, this allows a great deal of flexibility in how they are used. This design allows us to input the set of files containing the data needed, the access layer functions do the work of locating and correlating them.

2. Structure Definitions

The following are brief descriptions of the various structures used by this library and their role in the design.

2.1. Data Product

This is the highest level structure. It combines a complete set of associated elements along with the file pointers into a single unit. Note that the files in the file control list can be of the same type or of different types, it doesn't matter.

- Number of elements (data & keyword),
- Pointer to File Control structures,
- Pointers to element linked lists.

2.2. Data Element

This structure completely defines a data element. The element pointers allow elements to be organized by unique file/extension sets.

- Element common name (used by code),
- File Information structure,
- Column number and name (in file),
- Element description fields:
  format (TFORM), units (TUNIT), type (TTYPE), description,
- Dimensionality of element,
- Pointer to element value,
- Column number of time field associated with the element,
- Reference time of current value,
- Actual time of current and next record in file,
- Column number of data "quality" field,
- Data quality value associated with the record,
- Read/Write access control flag,
- Pointer to next data element of same set,
- Pointer to next set of data elements.

2.3. File Information

This structure contains the necessary information about individual file extensions. Each element contains a link to the file information structure appropriate for that element.

- Pointer to file list member,
- Extension description fields:
  number, name, HDU type,
- Current row for reading/writing,
- Number of element values in file (i.e. Number of rows in table).

2.4. File Control

The file control structure is the means by which the access layer functions manipulate the input set of files. It contains pointers to organize the files by type (defined by the CONTENT keyword). The "in_use" variable is used to decide when to open/close individual files.

- Pointer for FITS file,
- File name, type, access mode,
- Number of elements currently using the file,
- Pointer to next type of file,

- Pointer to next file of same type.

3. Implementation Overview

3.1. Data Product

In our implementation, there is a master template file which defines the data elements used by the various tools. These definitions match the software expectations for the data elements. The elements are organized in the template by a data product tag.

A function in the access layer library interprets this template, creates the specified elements and returns the linked set of data elements. Individual tools call this function for each input, specifying which data product is to be defined. In the case where inputs are uncertain and several different files must be provided together to supply the necessary input, the function is called in a way that combines several data product tags into a single linked set. The total number of keyword and data elements defined are stored in the data product fields

An alternate implementation would be to allow the data files themselves to define the data elements. However, you need to be confident that the data type will be compatible with the tool expectations.

3.2. File Information

Each data element contains a pointer to a file information structure. This structure contains the basic information about where to find the element. It is divided into two parts. The file control structure, described below, contains the highest level information about the file. The rest of the fields are items that are shared by elements found in the same file extension. These include the extension number, name and type, the current row of the file that is being accessed for this element, and the total number of rows this extension contains. If the extension is an image, these fields take on slightly different meanings.

3.3. File Control

The file control structure contains the highest level information about the data file. It contains the file pointer, file name and type. The file type is defined by the CONTENT keyword of the input data files. If multiple input files are provided, they are sorted by type using the next_file and next_type pointers. In this way, a set of files for each file type is created. All elements from a common file will share the same file control pointer. The "in_use" field of the structure tells the access layer how many elements are currently accessing this file. It is used to decide when to close out each file. As the data elements are read, they will move through the files of the same type until the end of all data is reached.

3.4. Time Handling

Time is a special beast. There is no data element for time, instead, it is a field in each data element. In fact, it is three fields. The data element structure has three time related fields, ref_time, curr_time, and next_time. Since this access layer must deal with inputs at differing sample rates, these fields help to correlate the data values.

The ref_time field holds the reference time for the elements. This is the time that is currently being considered in the program. All data element values should be valid for this reference time. The actual time of the current data record is stored in the curr_time field. The next_time field stores the time of the next data record. For table elements, these values will refer to the time column in the table. If no time column is found, or if the element is from an image extension, then the files TSTART and TSTOP keywords define the current and next time field values.

With this design, any number of data access schemes can be defined. Our primary algorithm steps through the data sequentially. When we request the next set of data values from the access layer, the code looks at all current elements and defines ref_time as the minimum value of the next_time fields. It then goes through all elements and reads a record from the data file for any element whose next_time is less than or equal to this reference time. The next_time value is shifted to the curr_time field and the time of the next record for that element is read. Elements that are not read because the current record is still valid for the reference time are repeated.

3.5. Data Element Handling

Once the data elements are defined with the template, the individual tools set the access privileges for each element that it wants to use. When the input data is opened, the first file of each type is searched for the elements to be accessed. As each element is found, the file information and, if appropriate, column fields are set. The elements are then sorted into sets that belong to common file types and extensions. This helps to optimize file access since we can read/write all elements from a common file/extension together instead of moving haphazardly between files and extensions.

Most data element fields are manipulated with a set of accessory functions provided in the library. These functions set or return the various field values. This allows the data element structure definition to be modified without having to change application tools.

The value field of the data element structure is simply a void pointer. Other fields in the structure define the size, type and dimensionality of the value. The value field is a pointer to the memory block containing the actual values. The calling routine must use this information to cast the value to a usable form. It is also the calling routines responsibility to allocate the memory block for the value.

4. Conclusion

This data access layer has allowed us to develop tools that are far less sensitive to input changes than would normally be possible. We can absorb changes to the number of inputs, the structure of the inputs, and even simple data type changes without the need to modify code.

Acknowledgments. This project is supported by the Chandra X-ray Center under NASA contract NAS8-39073.