Astronomical Data Analysis Software and Systems X
ASP Conference Series, Vol. 238, 2001
F. R. Harnden Jr., F. A. Primini, and H. E. Payne, eds.

Funtools: An Experiment with Minimal Buy-in Software

Eric Mandel, Stephen S. Murray, John Roll

Smithsonian Astrophysical Observatory, Cambridge, MA 02138

Abstract. Minimal buy-in software seeks to hide from its users the complexity of complex code. This means striking a balance between the extremes of full functionality (in which one can do everything, but it is hard to do anything in particular) and naive simplicity (in which it is easy to do the obvious things, but one can't do anything interesting). Minimal buy-in acknowledges that design decisions must be made up-front in order to achieve this balance.

Reported here are recent efforts to explore minimal buy-in software through the implementation of Funtools, a small suite of FITS library routines and analysis programs that attempts to hide the complexity of FITS coding. In particular, the use of "natural order" in the Funtools design enables it to "do the right thing" automatically.

1. Introduction

The Funtools[1] project arose out of conversations with astronomers about the decline in their software development efforts over the past decade. A stated reason for this decline is that it takes too much effort to master one of the existing FITS libraries simply in order to write a few analysis programs. This problem is exacerbated by the fact that astronomers typically develop new programs only occasionally, and the long interval between coding efforts often necessitates re-learning the FITS interfaces.

The goal was to develop a minimal buy-in FITS library for researchers who are occasional (but serious) coders. In this case, "minimal buy-in" meant "easy to learn, easy to use, and easy to re-learn next month". Conversations with astronomers interested in writing code indicated that this goal could be achieved by emphasizing two essential capabilities. The first was the ability to write FITS programs without knowing much about FITS, i.e., without having to deal with the arcane rules for generating a properly formatted FITS file. The second was to support the use of already-familiar C/Unix facilities, especially C structs and Unix stdio. Taken together, these two capabilities would allow researchers to leverage their existing programming expertise while minimizing the need to learn new and complex coding rules.

For example, the authors' group at SAO pursues research in X-ray astronomy, where the primary data are stored as rows of photon events in FITS binary

[1]An acronym for Fits Users Need TOOLS

tables. Each row consists of information about a single photon event, e.g., position, energy, and arrival time. In order to process photons in an X-ray table, one typically reads the events from stdin into an array of C records, manipulates these event records and then writes the results to stdout:

```
while( (nev=fread(ebuf, sizeof(EvRec), MAXEV, stdin)) ){
  for(i=0; i<nev; i++){
    ev = ebuf + i;
    if( ev->pha == XXX ) ev->energy = NewPHA(ev->pha);
  }
  fwrite(ebuf, sizeof(EvRec), nev, stdout);
}
```

In essence, the goal of the Funtools effort was to implement this code fragment as simply as possible for FITS binary tables.

2. Design Approach

The Funtools design approach was to *minimize the number of public subroutines* in its library. This approach was based on the hypothesis that a few carefully crafted routines would be easy to learn, easy to remember, and easy to use by occasional coders. The aim was to make it easier to "see the forest for the trees" by minimizing the number of trees.

The implications of this approach were two-fold. Firstly, the routines would have to be useful in their basic operation, while containing appropriate hooks to activate more sophisticated functionality. Routines would not be automatically added to the library in order to extend functionality. Instead, optional keyword specifiers were added to the calling sequence of basic routines, taking care to avoid making those routines overly complex along the way.

Secondly, it was decided that routines should act on behalf of the coder, especially with regard to FITS formatting issues (e.g., header generation and extension padding). Since proper formatting of FITS files is one of the most difficult and tedious parts of FITS programming, it is very desirable to have Funtools provide this service automatically. For example, when image or binary table data is written, the associated headers should be generated automatically as needed. In doing so, great care must be taken to ensure that all necessary parameters (including user-specified parameters) are correctly incorporated into the header. Designing such automatic services required maintaining the "state" of processing for a given FITS file, so that the library could "do the right thing" on behalf of the user.

Adhering to these design principles required the imposition on coders of a few rules of "natural order". Of course, order is important for any I/O library: one cannot read a file before it is opened or after it is closed. But in the current case, we decided to make this concept explicit. For example, in order to ensure that headers are automatically and properly generated when an image is written to a FITS extension using stdio, all parameters are required to be written before outputting the image data itself. It was hoped that users would accept such rules if they resulted in services being performed automatically and correctly. Examples of "natural order" are discussed below.

3. Implementation

The implementation strategy was centered on perfecting the code needed by a select few image and table processing algorithms, including the canonical X-ray analysis task: for each X-ray event (row in a binary table), read selected input columns into user space, modify the value of one or more of these columns, and output the results by merging new value(s) with the original input columns. The Funtools library implements this standard X-ray event-processing algorithm in a simple and straightforward manner:

```
typedef struct eventrec{ int pha, energy; } *Ev, EvRec;

ifun = FunOpen("in.fit[EV+,pha=5:9||pha==pi]", "rc", NULL);
ofun = FunOpen("out.fits", "w",  ifun);

FunColumnSelect(ifun, sizeof(EvRec), "merge=update",
   "pha",    "J", "r",  FUN_OFFSET(Ev, pha),
   "energy", "J", "rw", FUN_OFFSET(Ev, energy), NULL);

while( (ebuf=(Ev)FunTableRowGet(ifun, NULL, MAXEV, NULL, &nev)) ){
  for(i=0; i<nev; i++){
    ev = ebuf + i;
    if( ev->pha == XXX ) ev->energy = NewPHA(ev->pha);
  }
  FunTableRowPut(ofun, ebuf, nev, 0, NULL); free(ebuf);
}
FunClose(ofun); FunClose(ifun);
```

The Funopen() routine opens the specified FITS image or binary table extension and sets up the required column and spatial region filters. As shown above, the "output" call can be passed an input handle (argument 3) to inherit parameters and other information from the input file. In addition, if the input file is opened in "copy" mode ("c" in argument 2), the user can automatically copy the input extensions to the output file by appending "+" to the extension name ("EV+" in the example). The "natural order" rule here is that the input file must be opened before the output file.

The heart of Funtools processing for binary tables (i.e., X-ray events) is the FunColumnSelect() routine, which specifies how to read columns into a user-defined C struct and/or how to write columns to an output file. Named columns (arguments 4, 8, etc.) are read into the record structure offsets (arguments 7, 11, etc.) They are automatically converted from the FITS data type to the user-specified data type (arguments 5, 9, etc.) In addition, if the output file has inherited the input handle, then the output file also will know about columns having "w" mode. Finally, the optional "merge=[update/replace/append]" string (argument 3) specifies that processed columns will be merged with the original input columns on output. Thus, the same subroutine can be used to set up reading, writing, and/or merging of named columns.

Once columns from the binary table have been selected for processing, the routines FunTableRowGet() and FunTableRowPut() are used to get and put

rows. (Similar routines are available for image processing.) FunTableRowGet() reads event rows and stores the selected columns into an array of structs. Buffer space is allocated on the fly if argument 2 is NULL. FunTableRowPut() writes the event rows to the output file. Note that FITS headers are generated and written automatically as needed. To support these automatic services, the natural order rule is that get and put calls must be alternated.

Finally, the FunClose() routine is called to flush and close Funtools files. Note that FITS extension padding will be added automatically as needed (although one can call FunFlush() explicitly). Also note that the remaining input extensions are copied automatically to the output if in "copy" mode. To support these services, the natural order rule is to close the output file before the input file(s) to copy remaining extensions.

In Funtools, much of the complexity of dealing with FITS is hidden. FITS headers and extension padding are written automatically as needed. Output files easily inherit input parameters and other extension information. Copy of input extensions is specified easily on the command line. The price paid for these automatic services is the imposition of some rules of "natural order," although options are available for cases where coders are forced to violate these rules. For example, if input files must be closed before output files, coders can call FunFlush() beforehand to copy the remaining input files explicitly.

4. Discussion

Initial response to the Funtools library has been very positive, with researchers reporting that they often can write programs of considerable complexity in less than a day. Indeed, the early success of Funtools naturally leads to the question of how far a library of this sort can go. Are there limits to its functionality, beyond which a more traditional (i.e., complex) library is needed? Do the rules of "natural order" eventually impose unacceptable restrictions on the library or are they no more than common sense restrictions that everyone can follow? These and other issues can be explored during the further development of Funtools.

Funtools is available at SAO/HEAD R&D Group page.[2] It has been ported to Sun/Solaris, Linux, Dec Alpha, SGI, and Windows (using Cygwin). The Funtools library supports uniform access to FITS tables and images, and to non-FITS arrays and raw event lists. A number of sample programs are also offered; for example, funcnts calculates the background-subtracted image counts in user-specified regions. As always, suggestions and comments are welcome.

Acknowledgments. This work was performed in large part under a grant from NASA's Applied Information System Research Program (NAG5-9484), with support from the (Chandra) High Resolution Camera (NAS8-38248) and the Chandra X-ray Science Center (NAS8-39073). We wish to thank Leon VanSpeybroeck and Francis A. Primini for many helpful discussions.

[2] `http://hea-www.harvard.edu/RD`