The "simplifits" library

L.Chiappetti - IFCTR

Draft 0 - Fri, 22 Mar 1996

1 Introduction

The `simplifits` library is a very small library of Fortran subroutines which will be used by the ExDH simulator Science Console for the production of the event and ancillary files in FITS format (EPIC-EST-SP-004). According to the considerations in section 4 of EPIC-EST-SP-005, the routines do not pretend to be general enough, but will support only *a few limited, simple* cases (which however are the ones really needed !) :

- a file shall contain only one complete HDU, i.e. one either has a valid primary array (basic image HDU) and no extensions, or one has a primary HDU without a data array, followed by a *single* `BINTABLE` extension.
- the case of `IMAGE` extensions could be considered in the future.
- in case of extensions, the primary HDU shall contain only the mandatory keywords, all other header keywords shall go only in the (only) extension header.
- the number of header keywords shall be fixed at file creation time. Their content can however be updated later with the limitations described below.

`simplifits` currently consists of just 435 lines of Fortran (of which 210 lines of code), and fulfills most of the needs of the EPIC data archiving (support to images is still missing). This is therefore slimmer and easy to manage than a general purpose package like FITSIO 3.21 (18843 lines of Fortran of which 13063 of code).

The `simplifits` routines are provisionally imbedded in the source of the `processor.f` program, but are intended to be moved to a library later.

2 Typical sequence of actions

The `simplifits` routines are designed to be used to create concurrently a number n of different files (typically indexed by the HBR on which data are received, and typically one main [event or image] file and one ancillary science file), therefore they mantain n internal buffers for the n files.

n is currently 20, with a convention that n=1-7 is reserved to HBR 1-7 main science file, n=11-17 is reserved to HBR 1-7 ancillary files, and the other are spare.

Therefore the typical program shall proceed according to the following sequence :

- **open** (in any order) the main and ancillary files for a choice of HBRs (1 to 7)
- **populate the header** of each file in turn. This operation is subject to the following restrictions:
  - a separate call is necessary for each keyword
  - all the keywords of the header of one file shall be defined exactly in the sequence in which they are intended to appear in the file.
  - namely if there is a primary and an extension header, the primary header shall be written first and the extension next. Both shall be terminated by an `END` keyword.
  - if the value of a keyword is not known, it is allowed to use a dummy value for it.
  - it is not allowed to interleave setting keywords of different files, each one file's header must be completely defined until END before passing to the next file
  - there is no limitation on the order in which entire files are processed.

- **populate the data array**. This can be done either :
  - **filling an entire image** array at once

  or
  - **constructing** manually **one binary table row** (logical record) and
  - **adding the record** to the data area
  - records *within a file* are added sequentially
  - but records of *different* files can be freely interleaved
  - **flush the data area** when no more records need to be added, it is necessary to ensure explicitly that any intermediate data is written to disk.

- **update the header**. This is subject to similar limitations as indicated above for the header creation :
  - a separate call is necessary for each keyword to be updated to its final value
  - it is not necessary to touch keywords which are unchanged
  - it is not allowed to interleave modifying keywords of different files, each one file's header must be all done before passing to the next file
  - there is no limitation on the order in which entire files are processed.
  - there is no limitation on the order in which keywords of the header of one file are updated, but for efficiency reasons it is recommended to do this in the sequence in which they were created
  - it is not allowed to add new keywords.

- **close** the files. This operation is currently done automatically at the end of the program. There is no dedicated call. In the future it might be necessary to define a special call if one intends to release the buffer of one HBR for reuse.

Not all the above operations are done directly with `simplifits` calls. The library provides only elementary calls. Some of the higher level operations must be done within the calling program (e.g. writing high level routines). We provided below *examples* of the higher level procedure ("*outer layer*") and *calling sequences* for the `simplifits` routines ("*inner layer*:").

3 Outer layer

3.1 opening a file

The current `processor` program contains a `FITSINIT(HBR)` routine which will encode the names of the main and ancillary science file according to EPIC-EST-SP-004, and calls the `simplifits` routine `FITSOPEN` to actually open it.

Of course a different name encoding will be required for different modes.

3.2 populating the header

The same routine `FITSINIT` also defines the entire header for both files, namely in this order :

all the keywords in the primary header of the main file
all the keywords in the primary header of the ancillary file
all the keywords in the extension header of the main file
all the keywords in the extension header of the ancillary file

Each header is done by a sequence of call to the `FITSADDx` routines (where `x` specifies whether the keyword is logical, integer, real or string), terminated by a call to `FITSADDE` to write the `END`. Note that it is allowed to change file after each `FITSADDE` call !

3.3 adding image data

Currently not yet supported.

3.4 creating and adding a table row

This is the most complex operation, and currently it is split in two parts. One is the creation of the record, and the other one is the insertion of it into the FITS block. Currently the first one is done separately in the outer layer, and `simplifits` provides only the `FITSDATA` call for the second one.
Let's assume for the sake of argument that a given file format has "rows" with the following layout :

TFORM1	1J	first column A1 is 32-bit integer
TFORM2	1I	second column A2 is 16-bit integer
TFORM3	1J	third column A3 is 32-bit integer
TFORM4	1I	fourth column A4 is 16-bit integer
TFORM5	1I	and so fifth A5

which will require a 14-byte record.

The idea is to have *in the calling program* a dedicated routine for each record layout (so effectively for each of the different file formats in EPIC-EST-SP-004), which receives as arguments the `hbr` and the elements `A1,A2,A3,A4,A5`. For the sake of simplicity, these do not need to exist in the caller as `INTEGER*2` or `INTEGER*4`, but can be all default `INTEGER`. Any conversion will be done within the routine.
The routines `FITS_MEI` and `FITS_MAI` in `processor` follow this pattern and cater for MOS Imaging event and ancillary file respectively.

Continuing with the example above, one shall do the following in the outer layer routine :

declare a `CHARACTER*14 databuf`.
put it in equivalence with an `INTEGER*2 ibuf(7)`
put it in equivalence with an `INTEGER*4 jbuf(3)`

Now some of the arguments can be inserted directly into the buffer as :
`jbuf(1)=A1`
`ibuf(3)=A2`
`ibuf(6..7)=A4` and `A5`

but this cannot be done for A3 since it is misaligned w.r.t 32-bit boundaries, therefore to overcome this a portable way is to have a CHARACTER*4 C4 in equivalence with an INTEGER I4.
then assign `I4=A3`
and then place into buffer with `databuf(7:10)=C4`

at this point call the inner layer routine with `CALL FITSDATA(hbr,databuf,14)`

The above is enough in the case.the internal representation of the machine one is running on has the same endianness required by FITS.(this is the case of HP-UX and SunOS). On the contrary on machines with reverse endianness (DEC Ultrix, DEC OSF) it is care of the caller to do the *appropriate* byteswapping of `databuf` before calling `FITSDATA`. The routines `FITS_MEI` and `FITS_MAI` in `processor` contain an example of how to do it.
Note that while elements of `ibuf` and (32-bit aligned) elements of `jbuf` can be swapped directly, misaligned 32-bit word must be swapped before insertion in `databuf`.

3.5 flushing the data area

The current `processor` program contains a `FITSEND(HBR)` routine which as first thing does, once for each file, a "special" `FITSDATA` call to ensure all unwritten data is writtten to disk.

3.6 updating the header

The same routine FITSEND contains for each file a sequence of call to the `FITSUPDx` routines (where `x` specifies whether the keyword is logical, integer, real or string), to update any keyword whose value is known only at the end of the processing.

3.7 closing the file

There is currently no need of a special call to do this.

4 Inner layer

The `simplifits` routines share `COMMON` information, initialized by `BLOCK DATA` routine `FITSBLK`, namely :

an array of 20 logical units
an array of 20 record pointers
an array of 20 byte pointer inside the record
a single FITS block buffer for headers (initialized to blanks)

while the `FITSDATA` routines contains an array of 20 FITS block buffers (initialized to zero) for binary table data.

All such 20 items are indexed by an `HBR` argument, which conventionally assumes the values 1-7 for the main files and 11-17 for the ancillary files of HBRs 1-7. The remaining locations are spare.
The calling program normally has no need to know or use the logical unit associated to a given file.

4.1 opening a file

The opening of a file called `filename`, associated to a given `hbr` argument (1-7 or 11-17) is obtained with the following call (this also returns for information only the logical unit `lu`) :

```
CALL FITSOPEN(hbr,filename,lu)
```

Currently an existing file of same name is overwritten. On the contrary no file can be associated to a given `hbr` if this is already allocated. The file is opened via a `Z_OPEN` XAS call. Error checking must be made in the calling program using the XAS `VOSERROR` call.

Note that provisionally `FITSOPEN` also calls `FITSDATA` to initialize the FITS data block buffer to zero. This could be done via an implied-do `DATA` statement, but this is extremely slow during compilation, therefore at least during development the current arrangement is preferred.

4.2 populating the header

The following calls are used to add a single keyword at a time to the primary or extension header. They keep internally track of the current record. All calls have the same sequence, where `keyword` is a character string with the keyword name, and `value` is the value (in the appropriate type) :

```
CALL FITSADDL(hbr,keyword,lvalue)    for a LOGICAL value
CALL FITSADDI(hbr,keyword,ivalue)    for an INTEGER value
```

```
CALL FITSADDR(hbr,keyword,rvalue)    for a REAL value
CALL FITSADDC(hbr,keyword,cvalue)    for a CHARACTER value

CALL FITSADDE(hbr)
```

is instead the call used to write the `END` keyword and flush to disk the last block of the header.

4.3 adding image data

Currently not yet supported

4.4 adding a table row

A logical record containing a table row, kept in a `CHARACTER*buflen` buffer `databuf`, and created by the outer layer of the program already with the right FITS byte order, can be "virtually" written to disk with the call :

```
CALL FITSDATA(hbr,databuf,buflen)
```

The call inserts `databuf` in the current FITS block for the given `hbr`, and, whenever the block is full, writes it to disk, and reinitializes the block. It also caters for the case when `databuf` spans across different FITS blocks.

The routine also performs the function described in the next section.

4.5 flushing the data area

This is obtained by issuing a `FITSDATA` call with a zero lenght, i.e. doing

```
CALL FITSDATA(hbr,0,0)
```

which results in the current FITS block being written to disk, if it is partially full. If it is empty it is not written to disk. This call is also used by `FITSOPEN` to initialize the FITS block to zero at beginning of the program (in such case nothing is written to disk).

4.6 updating the header

The following calls are used to modify a single header keyword. The routines search internally the primary and extension header (only if consecutive !), starting from the current position to the end, and, if necessary, back from beginning to end. If a keyword is found, its value is modified and the FITS block is written to disk (this is not extremely efficient, one might improve it writing a modified block only when a change of block is implied). All calls have the same sequence as the `FITSADDx` calls :

```
CALL FITSUPDI(hbr,keyword,ivalue)    for an INTEGER value
CALL FITSUPDR(hbr,keyword,rvalue)    for a REAL value
CALL FITSUPDC(hbr,keyword,cvalue)    for a CHARACTER value
```

There is currently no call to modify a `LOGICAL` value.

4.7 closing the file

No call currently exist or is needed. However a `FITSCLOSE` call might be necessary to deallocate the `COMMON` elements associated to an `hbr` if re-use in the same program would be intended.