# SFITSIO

**A next−generation FITS I/O library**
for C and C++ programmers

SFITSIO is a new library written from full scratch. The concept is to provide script-like APIs which minimize programmer's efforts for FITS I/O. It enables you to write intuitively your codes for FITS. SFITSIO will free you from looking through the manuals when writing codes for FITS I/O.

Web page: `http://www.ir.isas.jaxa.jp/~cyamauch/sli/`


***Chisato Yamauchi***
**`cyamauch@ir.isas.jaxa.jp`**
*Center of Science−satellite Operation and Data Archive,*
*Institute of Space and Astronautical Science,*
*Japan Aerospace Exploration Agency*


## 1 Status

- Stable version 1.0 has been released in September 2009.
  Linux, MacOSX, FreeBSD, Solaris and Cygwin are supported (64-bit ready).
- SFITSIO is supported by C-SODA/ISAS/JAXA.

## 2 Concepts

- SFITSIO is designed for C programmers and C++ programmers. Having some experience of C is enough to use SFITSIO. The manual of SFITSIO is written in the C-language styles.
- SFITSIO is developed with very simple object-oriented technique to provide convenient APIs like script languages.
- SFITSIO is written in C++, but uses neither STL nor C++-specific manners (such as overuse of "<<" operator). You do not have to learn them to use SFITSIO. Of course, you can use
  `printf(...);`
  as usual, and SFITSIO also has many APIs in printf() style.

## 3 Usefulness

- Extremely simple and easy APIs minimize the volume of your codes for FITS.
- Free from the memory managements. SFITSIO automatically allocates and frees required memory.
- Free from looking through the manuals. APIs of SFITSIO represent the structures of FITS files as-is, so you hardly forget the specification of the APIs if you write once.
- Compression and decompression of files (`.gz` or `.bz2`) are automatic. You can access both compressed and plain FITS files even on remote HTTP or FTP servers without any special APIs.
- Provides image APIs to copy, paste, add, subtract, multiply and divide any rectangular regions (user functions can be defined).
- Provides table APIs to manipulate columns and rows.
- Combination with WCSLIB is straightforward.

## 4 Support of FITS standards

- Basically all HDU types are supported.

| Image | Group | Ascii Table | Binary Table |
|---|---|---|---|
| OK | - | OK | OK |

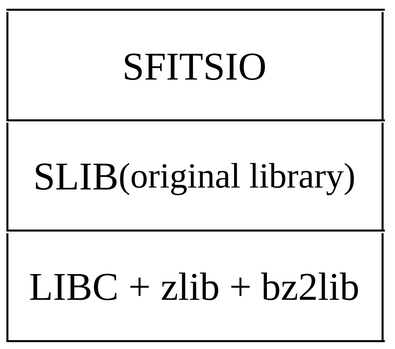
- Multi-dimensional array of binary table is supported.
- Variable length array of binary table will be supported in near future.

## 5 Support of non-standard extensions

- Long header value using `CONTINUE` keyword is supported.
- Supports long keyword of maximum 75 characters in the header.
- Number of columns of the Ascii/Binary table can exceed 999.

## 6 Design of implementation

- Very simple layer structure.




SFITSIO is built based on the "SLIB" developed by me. SLIB appends many useful APIs like script languages to C language. SLIB provides classes to handle various streams (stdio, compressed stream, stream via network, etc.), strings and multi-dimensional array.

SFITSIO always accesses LIBC/zlib/bz2lib functions through SLIB, which means that the internal source of SFITSIO does not include codes for compression/decompression.

APIs of SLIB are public. Using SLIB, you can easily create original I/O libraries for other file formats which support the compression and the network without your knowledge about raw APIs.

## 7 Examples of user codes

- Read a remote file *http://foo/myfits.fits.bz2* and write *myfits.fits.gz* locally.

```
fitscc fits;
fits.read_stream("http://foo/myfits.fits.bz2");
fits.write_stream("myfits.fits.gz");
```

- Display the header value of the `TELESCOP` and `EQUINOX` keyword.

```
printf("TELESCOP = %s  EQUINOX = %f\n",
       fits.image("Primary").header("TELESCOP").svalue(),
       fits.image("Primary").header("EQUINOX").dvalue());
```

- Create new image FITS and save it to *myimage.fits.gz*.

```
fitscc fits;
fits.append_image("Primary",0, FITS::DOUBLE_T,
                  1024, 1024); /* size of pixels */
fits.writef_stream("myimage.fits%s", ".gz");
```

- Read and write pixel values applied `BZERO` and `BSCALE`.

```
fits_image &primary = fits.image("Primary");
double val = primary.dvalue(x0,y0,z0);  /* read */
primary.assign(val, x1,y1,z1);          /* write */
```

- Display all rows in `TDISP`*n* formats of all columns and elements of a table in the *CATALOG* HDU stored in *mytable.fits.gz*. This code is applicable to both Ascii and Binary tables.

```
fitscc fits;
long i, j, k, n_elem;
fits.read_stream("mytable.fits.gz");
fits_table &bte = fits.table("CATALOG");
for ( i=0 ; i < bte.row_length() ; i++ ) {
    for ( j=0 ; j < bte.col_length() ; j++ ) {
        n_elem = bte.col(j).elem_length();
        for ( k=0 ; k < n_elem ; k++ ) {
            printf("%s ",bte.col(j).svalue(i,k));
        }
    }
    printf("\n");
}
```