FITSIO Documentation

Release 0.8.0

FITSIO.jl Contributors

Jun 20, 2017

Contents

A Julia package for reading and writing Flexible Image Transport System (FITS) files, based on the cfitsio library. The interface is inspired by Erin Sheldon's fitsio Python package.

CHAPTER 1

Install

```
julia> Pkg.add("FITSIO")
```

On linux or OS X, if it isn't already installed on your system, the cfitsio library is automatically downloaded and compiled (in your Julia packages directory). On Windows, a compiled dll will be downloaded.

CHAPTER 2

Usage

Open an existing file for reading:

```
julia> using FITSIO

julia> f = FITS("file.fits")
file: file.fits
mode: r
extnum exttype         extname
1      image_hdu
2      binary_table
```

(At the REPL, information about the file contents is shown.)

A FITS file consists of one or more header-data units (HDUs), concatenated one after the other. The `FITS` object therefore is represented as a collection of these HDUs.

Get information about the first HDU:

```
julia> f[1]  # get the first extension
file: file.fits
extension: 1
type: IMAGE
image info:
  bitpix: -64
  size: (800,800)
```

Iterate over HDUs in the file:

```
julia> for hdu in f; println(typeof(hdu)); end
ImageHDU
TableHDU
```

Each HDU can contain image data, or table data (either binary or ASCII-formatted). For image extensions, get the size of the image without reading it:

```
julia> ndims(f[1])
2

julia> size(f[1])
(800,800)

julia> size(f[1], 2)
800
```

Read an image from disk:

```
julia> data = read(f[1]);  # read an image from disk

julia> data = read(f[1], :, 790:end);  # read just a subset of image
```

Show info about a binary table:

```
julia> f[2]
file: file.fits
extension: 2
type: BINARY TABLE
rows: 20
columns:
    col2 (5A)
    col1 (1K)
```

Read a column from the table:

```
julia> data = read(f[2], "col1")
```

Read the entire header into memory and get values from it:

```
julia> header = read_header(f[1]);  # read the entire header from disk

julia> length(header)  # total number of records in header
17

julia> haskey(header, "NAXIS1")  # check if a key exists
true

julia> header["NAXIS1"]  # get value by keyword
800

julia> header[4]  # get value by position
800

julia> get_comment(header, "NAXIS")  # get comment for a given keyword
"length of data axis 1"
```

Read just a single header record without reading the entire header:

```
julia> read_key(f[1], 4)  # by position
("NAXIS1",800,"length of data axis 1")

julia> read_key(f[1], "NAXIS1")  # read by keyword
(800,"length of data axis 1")
```

Manipulate a header in memory:

```
julia> header["NEWKEY"] = 10  # change or add a keyword

julia> set_comment!(header, "NEWKEY", "this is a comment")
```

Close the file:

```
julia> close(f)
```

(`FITS` objects are also closed automatically when garbage collected.)

Open a new file for writing:

```
julia> f = FITS("newfile.fits", "w");
```

The second argument can be `"r"` (read-only; default), `"r+"` (read-write) or `"w"` (write). In "write" mode, any existing file of the same name is overwritten.

Write an image to the file:

```
julia> data = reshape([1:100], 5, 20);

julia> write(f, data)  # Write a new image extension with the data
```

To write some header keywords in the new extension, pass a `FITSHeader` instance as a keyword: `write(f, data; header=header)`

Write a table to the file:

```
julia> data = ["col1"=>[1., 2., 3.], "col2"=>[1, 2, 3]];

julia> write(f, data)  # write a new binary table to a new extension
```

CHAPTER 3

Reference

API reference

File operations

FITS (*filename::String*, *mode::String="r"*)
Open or create a FITS file. `mode` can be one of `"r"` (read-only), `"r+"` (read-write) or `"w"` (write). In "write" mode, any existing file of the same name is overwritten.

A `FITS` object is a collection of "Header-Data Units" (HDUs) and supports the following operations:

- `f[i]` Return the `i`-th HDU.
- `f[name]` or `f[name, ver]` Return the HDU containing the given the given EXTNAME (or HDUNAME) keyword (an ASCIIString), and optionally the given EXTVER (or HDUVER) number (an Integer).
- Iteration:

```
for hdu in f
    ...
end
```

length (*f::FITS*)
Number of HDUs in the file.

close (*f::FITS*)
Close the file. Subsequent attempts to operate on `f` will result in an error. `FITS` objects are also automatically closed when they are garbage collected.

Header operations

read_header (*hdu*)
Read the entire header from the given HDU and return a `FITSHeader` object. The value of each header record

is parsed as `Int`, `Float64`, `ASCIIString`, `Bool` or `nothing` according to the FITS standard. (If the value cannot be parsed according to the FITS standard, the value is stored as the raw unparsed `ASCIIString`.)

`read_header` (*hdu*, *ASCIIString*)
Read the entire header from the given HDU as a single string.

`FITSHeader` (*keys*, *values*, *comments*)
Create a `FITSHeader` from arrays of keywords, values and comments. This type partially implements the Associative interface:

- `length(hdr)` Number of records.
- `haskey(hdr)` Header keyword exists.
- `keys(hdr)` Array of keywords (not a copy).
- `values(hdr)` Array of values (not a copy).
- `hdr[key]` Get value based on keyword or index.
- `hdr[key] = value` Set value based on keyword or index.

Additionally, there are functions to get and set comments:

- `get_comment(hdr, key)` Get the comment based on keyword or index.
- `set_comment!(hdr, key, comment)` Set the comment baed on keyword or index.

`read_key` (*hdu*, *key*)
Read just the specified key and return a tuple of `(value, comment)`. The key can be either the index of the header record (Integer) or the header keyword (ASCIIString).

Image operations

`write` (*f::FITS*, *data::Array; header=nothing*, *name=nothing*, *ver=nothing*)
Add a new ImageHDU to the file. The following array element types are supported: `UInt8`, `Int8`, `UInt16`, `Int16`, `UInt32`, `Int32`, `Int64`, `Float32`, `Float64`. If a `FITSHeader` object is passed as the `header` keyword argument, the header will be added to the new HDU.

`read` (*hdu::ImageHDU*)
Read the entire image from disk.

`read` (*hdu::ImageHDU*, *range...*)
Read a subsection of the image from disk. E.g., `read(hdu, 1:20, 1:2:20)`.

`ndims` (*hdu::ImageHDU*)
Get number of image dimensions, without reading the image into memory.

`size` (*hdu::ImageHDU*)
Get image dimensions, without reading the image into memory.

`size` (*hdu::ImageHDU*, *i::Integer*)
Get `i`-th dimension.

`length` (*hdu::ImageHDU*)
Get total number of pixels in image (product of `size(hdu)`).

`copy_section` (*hdu::ImageHDU*, *dest::FITS*, *r::Range...*)
Copy a rectangular section of an image and write it to a new FITS primary image or image extension. The new image HDU is appended to the end of the destination file; all the keywords in the input image will be copied to the output image. The common WCS keywords will be updated if necessary to correspond to the coordinates of the section. Examples:

Copy the lower-left 200 x 200 pixel section of the image in `hdu` to an open file, `f`:

```
copy_section(hdu, f, 1:200, 1:200)
```

Same as above but only copy odd columns in y:

```
copy_section(hdu, f, 1:200, 1:2:200)
```

Table Operations

`write`(*f::FITS*, *data::Dict;* *hdutype=TableHDU*, *name=nothing*, *ver=nothing*, *header=nothing*, *units=nothing*, *varcols=nothing*)

Create a new table extension and write data to it. If the FITS file is currently empty then a dummy primary array will be created before appending the table extension to it. `data` should be a dictionary with ASCIIString keys (giving the column names) and Array values (giving data to write to each column). The following types are supported in binary tables: `Uint8`, `Int8`, `Uint16`, `Int16`, `Uint32`, `Int32`, `Int64`, `Float32`, `Float64`, `Complex64`, `Complex128`, `ASCIIString`, `Bool`.

Optional inputs:

- `hdutype`: Type of table extension to create. Can be either `TableHDU` (binary table) or `ASCIITableHDU` (ASCII table).
- `name`: Name of extension.
- `ver`: Version of extension (Int).
- `header`: FITSHeader instance to write to new extension.
- `units`: Dictionary mapping column name to units (as a string).
- `varcols`: An array giving the column names or column indicies to write as "variable-length columns".

Note: Variable length columns

Variable length columns allow a column's row entries to contain arrays of different lengths. They can potentially save diskspace when the rows of a column vary greatly in length, as the column data is all written to a contiguous heap area at the end of the table. Only column data of type `Vector{ASCIIString}` or types such as `Vector{Vector{UInt8}}` can be written as variable length columns. In the second case, ensure that the column data type is a *leaf type*. That is, the type cannot be `Vector{Vector{T}}`, which would be an array of arrays having potentially non-uniform element types (which would not be writable as a FITS table column).

`write`(*f::FITS*, *colnames*, *coldata;* *hdutype=TableHDU*, *name=nothing*, *ver=nothing*, *header=nothing*, *units=nothing*, *varcols=nothing*)

Same as `write(f::FITS, data::Dict; ...)` but providing column names and column data as a separate arrays. This is useful for specifying the order of the columns. Column names must be `Array{ASCIIString}` and column data must be an array of arrays.

`read`(*hdu*, *colname*)

Read a column as an array from the given table HDU.

The column name may contain wild card characters (`*`, `?`, or `#`). The `*` wild card character matches any sequence of characters (including zero characters) and the `?` character matches any single character. The `#` wildcard will match any consecutive string of decimal digits (0-9). The string must match a unique column.

Miscellaneous

`FITSIO.libcfitsio_version()` → VersionNumber
Return the version of the underlying CFITSIO library. E.g., `v"3.34.0"`.

Libcfitsio submodule

The `Libcfitsio` submodule provides an interface familiar to users of the CFITSIO C library. It can be used with

```
using FITSIO.Libcfitsio
```

The functions exported by this module operate on `FITSFile` objects. For the most part, they are thin wrappers around the CFITSIO routines of the same names. Typically, they:

- Convert from Julia types to C types as necessary.
- Check the returned `status` value and raise an appropriate exception if non-zero.

Warning: Note that these functions do not check if the file is still open before trying to access it. A segmentation fault can result from trying to operate on a closed file. (The main FITSIO interface always checks if the file is open before any operation.)

File access

fits_create_file (*filename::String*)
Create and open a new empty output `FITSFile`.

fits_clobber_file (*filename::String*)
Like fits_create_file, but overwrites `filename` if it exists.

fits_open_file (*filename::String*)
Open an existing data file.

fits_open_table (*filename::String*)
Open an existing data file (like *fits_open_file()*) and move to the first HDU containing either an ASCII or a binary table.

fits_open_image (*filename::String*)
Open an existing data file (like *fits_open_file()*) and move to the first HDU containing an image.

fits_open_data (*filename::String*)
Open an existing data file (like *fits_open_file()*) and move to the first HDU containing either an image or a table.

fits_close_file (*f::FITSFile*)
Close a previously opened FITS file.

fits_delete_file (*f::FITSFile*)
Close an opened FITS file (like *fits_close_file()*) and removes it from the disk.

fits_file_name (*f::FITSFile*)
Return the name of the file associated with object *f*.

HDU Routines

The functions described in this section allow to change the current HDU and to find their number and type. The following is a short example which shows how to use them:

```
num = fits_get_num_hdus(f)
println("Number of HDUs in the file: ", num)

for i = 1:num
    hdu_type = fits_movabs_hdu(f, i)
    println(i, ") hdu_type = ", hdu_type)
end
```

fits_get_num_hdus (*f::FITSFile*)
Return the number of HDUs in the file.

fits_movabs_hdu (*f::FITSFile*, *hduNum::Integer*)
Change the current HDU to the value specified by *hduNum*, and return a symbol describing the type of the HDU. Possible symbols are: `:image_hdu`, `:ascii_table`, or `:binary_table`.

The value of *hduNum* must range between 1 and the value returned by *fits_get_num_hdus()*.

fits_movrel_hdu (*f::FITSFile*, *hduNum::Integer*)
Change the current HDU by moving forward or backward by *hduNum* HDUs (positive means forward), and return the same as *fits_movabs_hdu()*.

fits_movnam_hdu (*f::FITSFile*, *extname::String*, *extver::Integer=0*, *hdu_type_int::Integer=-1*)
Change the current HDU by moving to the (first) HDU which has the specified extension type and EXTNAME and EXTVER keyword values (or HDUNAME and HDUVER keywords). If `extver` is 0 (the default) then the EXTVER keyword is ignored and the first HDU with a matching EXTNAME (or HDUNAME) keyword will be found. If `hdu_type_int` is -1 (the default) only the extname and extver values will be used to locate the correct extension. If no matching HDU is found in the file, the current HDU will remain unchanged.

Header Keyword Routines

fits_get_hdrspace (*f::FITSFile*) -> (*keysexist*, *morekeys*)
Return the number of existing keywords (not counting the END keyword) and the amount of space currently available for more keywords.

fits_read_keyword (*f::FITSFile*, *keyname::String*) -> (*value*, *comment*)
Return the specified keyword.

fits_read_record (*f::FITSFile*, *keynum::Int*) → String
Return the nth header record in the CHU. The first keyword in the header is at `keynum = 1`.

fits_read_keyn (*f::FITSFile*, *keynum::Int*) -> (*name*, *value*, *comment*)
Return the nth header record in the CHU. The first keyword in the header is at `keynum = 1`.

fits_write_key (*f::FITSFile*, *keyname::String*, *value*, *comment::String*)
Write a keyword of the appropriate data type into the CHU.

fits_write_record (*f::FITSFile*, *card::String*)
Write a user specified keyword record into the CHU.

fits_delete_record (*f::FITSFile*, *keynum::Int*)
Delete the keyword record at the specified index.

fits_delete_key (*f::FITSFile*, *keyname::String*)
Delete the keyword named `keyname`.

fits_hdr2str (*f::FITSFile*, *nocomments::Bool=false*)
Return the header of the CHDU as a string. If `nocomments` is `true`, comment cards are stripped from the output.

Primary Array Routines

fits_get_img_size (*f::FITSFile*)
Get the dimensions of the image.

fits_create_img (*f::FITSFile*, *t::Type*, *naxes::Vector{Int}*)
Create a new primary array or IMAGE extension with a specified data type and size.

fits_write_pix (*f::FITSFile*, *fpixel::Vector{Int}*, *nelements::Int*, *data::Array*)
Write pixels from *data* into the FITS file.

fits_read_pix (*f::FITSFile*, *fpixel::Vector{Int}*, *nelements::Int*, *data::Array*)
Read pixels from the FITS file into `data`.

Table Routines

To create ASCII/binary tables in a new HDU, the FITSIO.jl library provides two functions: `fits_create_ascii_table()` and `fits_create_binary_table()`. In general, one should pick the second as binary tables require less space on the disk and are more efficient to read and write. (Moreover, a few datatypes are not supported in ASCII tables). In order to create a table, the programmer must specify the characteristics of each column by passing an array of tuples. See the documentation of `fits_create_ascii_table()` for more details.

Here is an example:

```
f = fits_create_file("!new.fits")
coldefs = [("SPEED", "1D", "m/s"),
           ("MASS", "1E", "kg"),
           ("PARTICLE", "20A", "Name")]
fits_create_binary_tbl(f, 10, coldefs, "PARTICLE")
```

This example creates a table with room for 10 entries, each of them describing the characteristics of a particle: its speed, its mass, and its name (codified as a 20-character string).

fits_create_ascii_tbl (*f::FITSFile*, *numrows::Integer*, *coldefs::Array{ColumnDef}*, *extname::String*)
Append a new HDU containing an ASCII table. The table will have *numrows* rows (this parameter can be set to zero), each initialized with the default value. The columns are specified by the *coldefs* variable, which is an array of tuples. Each tuple must have three string fields:

1. The name of the column.
2. The data type and the repetition count. It must be a string made by a number (the repetition count) followed by a letter specifying the type (in the example above, `D` stands for *Float64*, `E` stands for Float32, `A` stands for `Char`). Refer to the CFITSIO documentation for more information about the syntax of this parameter.
3. The measure unit of this field. This is used only as a comment.

The value of *extname* sets the "extended name" of the table, i.e., a string that in some situations can be used to refer to the HDU itself.

Note that, unlike for binary tables, CFITSIO puts some limitations to the types that can be used in an ASCII table column. Refer to the CFITSIO manual for further information.

See also *`fits_create_binary_tbl()`* for a similar function which creates binary tables.

fits_create_binary_tbl (*f::FITSFile*, *numrows::Integer*, *coldefs::Array{ColumnDef}*, *extname::String*)
Append a new HDU containing a binary table. The meaning of the parameters is the same as in a call to `fits_create_ascii_tbl()`.

fits_get_coltype (*f::FITSFile*, *colnum::Integer*)
Provided that the current HDU contains either an ASCII or binary table, return information about the column at position `colnum` (counting from 1). Return is a tuple containing

- `typecode`: the CFITSIO integer type code of the column
- `repcount`: the repetition count for the column
- `width`: the width of an individual element

fits_insert_rows (*f::FITSFile*, *firstrow::Integer*, *nrows::Integer*)
Insert a number of rows equal to *nrows* after the row number *firstrow*. The elements in each row are initialized to their default value: you can modify them later using `fits_write_col()`.

Since the first row is at position 1, in order to insert rows *before* the first one *firstrow* must be equal to zero.

See also `fits_delete_rows()`.

fits_delete_rows (*f::FITSFile*, *firstrow::integer*, *nrows::Integer*)
Delete *nrows* rows, starting from the one at position *firstrow* (the first row has index 1).

See also `fits_insert_rows()`.

fits_read_col (*f::FITSFile*, *colnum::Integer*, *firstrow::Integer*, *firstelem::Integer*, *data::Array*)
Read data from one column of an ASCII/binary table and convert the data into the specified type *T*. The column number is specified by *colnum* (the first column has `colnum=1`). The elements to be read start from the row number *firstrow*; in case each cell contains more than one element (i.e., the "repetition count" of the field is greater than one), *firstelem* allows to specify which is the first element to be read. The overall number of elements is specified by the length of the array *data*, which at the end of the call will be filled with the elements read from the column.

fits_write_col (*f::FITSFile*, *colnum::Integer*, *firstrow::Integer*, *firstelem::Integer*, *data::Array*)
Write some data in one column of a ASCII/binary table. The column number is specified by *colnum* (the first column has `colnum=1`). The first element is written at the position *firstelem* within the row number *firstrow* (both the indexes start from one).

If there is no room for the elements, new rows will be created. (It is therefore useless to call `fits_insert_rows()` if you only need to *append* elements to the end of a table.)

Index