S-Lang CFITSIO Module Reference

John E. Davis, davis@space.mit.edu Apr 19, 2007

Contents

Chapter 1

Introduction

FITS `http://heasarc.gsfc.nasa.gov/docs/heasarc/fits.html` (Flexible Image Transport System) is a data format that is in widespread use by the astronomical community.

CFITSIO `http://heasarc.gsfc.nasa.gov/docs/software/fitsio/c/c_user/cfitsio.html` is a popular C library that interfaces to such files and provides support for all features of the format. Moreover CFITSIO supports a number of unofficial or proposed FITS conventions that are in widespread use.

This package makes use of the CFITSIO library allow one to manipulate FITS files from the **S-lang** interpreter. The package consists of two interfaces: a high level interface and a low level one. The low level interface is implemented as a module and is more or less a straightforward wrapping of the functions of the CFITSIO library. Functions from this interface are prefixed with an underscore to indicate that they are part of the low-level interface. The high level interface is written in **S-lang** and makes use of functions from the low level interface. While there is some overlap with the semantics of the CFITSIO library, the high level interface should be regarded as a separate interface to fits files.

To illustrate the difference between the two interfaces, consider the low-level `_fits_read_col` function and its high level counterpart `fits_read_col`. The low-level function reads a single column, specified via the column number, from a fits binary table and performs a minimal amount of error checking. In contrast, `fits_read_col` is a high level function that reads one or more columns, specified either as column numbers or named columns, from a table and does so in a way that takes into account the way CFITSIO performs buffering for maximum efficiency. Moreover, the high level function checks for the presence of TDIM keywords or columns to give the arrays it returns the proper dimensionality. If any errors occur, the function will throw an exception.

Chapter 2

The high-level interface

2.1 Overview

The high-level interface consists of a number of functions that are written in **S-lang** and are designed to take some of the tedium out of performing standard operations on fits files. To illustrate this point, consider the creation of a fits file with a binary table extension called called "COSXSINX":

```
variable data = struct { x, cosx, sinx };
data.x = [0:2*PI:0.01];
data.cosx = cos(data.x);
data.sinx = sin(data.x);
fits_write_binary_table ("foo.fits", "COSXSINX", data);
```

It can't get much easier than that!

2.2 Opening and Closing Files

In general the high-level functions take an argument that represents the fits file to be manipulated. It may be either an already open file pointer such as one returned by `fits_open_file`, or the name of a file to be opened. In the documentation for the functions, this fact is indicated by

```
Fits_File_Type or String_Type fd;
```

showing that the file descriptor may be either an open file pointer or a string.

If specified as a string, then a fits file of that name will be opened in a mode that is compatible with the operation being performed, with the current HDU (header-data unit) set to the first "most interesting" one. Here, "first most interesting" means the first HDU with a non-zero value for the NAXIS keyword.

For example, sometimes one simply wants to read some keywords from a file. In such a case it is not necessary to explicitly call `fits_open_file`. Rather simply pass the name of the file to the appropriate function:

```
(object, ra_targ, dec_targ)
   = fits_read_key ("foo.fits", "OBJECT", "RA_TARG", "DEC_TARG");
```

It may be necessary to specify which HDU should be used if the "first most interesting" one is not the desired HDU. The easiest way to do that is to specify the extension using CFITSIO's virtual file syntax, e.g.,

```
(object, ra_targ, dec_targ)
   = fits_read_key ("foo.fits+1", "OBJECT", "RA_TARG", "DEC_TARG");
(object, ra_targ, dec_targ)
   = fits_read_key ("foo.fits[EVENTS]", "OBJECT", "RA_TARG", "DEC_TARG");
```

If one is going to make a number of calls to functions in the high-level interface using the same file, then it is a good idea to explicitly open the file, e.g.,

```
fptr = fits_open_file ("foo.fits[EVENTS]", "w");
```

opens the file for both reading and writing and sets the current HDU to the "EVENTS" extension.

The object returned by the `fits_open_file` function is an object of type `Fits_File_Type`. It is automatically destroyed when it goes out of scope. When this happens, the fits file attached to it will be silently closed. Consider:

```
define write_image_to_file (file, img)
{
   variable fptr = fits_open_file ("new.fits", "c");
   fits_write_image (fptr, NULL, img, NULL, NULL);
   fits_write_date (fptr);
}
```

Here, the `write_image_to_file` function will create a new file called `new.fits` and write the specified image to the file. In this example, the file pointer object was not explicitly closed. Since the `fptr` variable goes out of scope when the function returns, the cfitsio module will silently close the file. While this is a convenient feature for many purposes, it is always better to explicitly close a file when it has been modified. The reason for this is that CFITSIO writes to internal buffers and then flushes those to the disk. Often some buffers will not get written to the disk until the file is closed. If the disk is full, or something else goes wrong then the file will not be properly closed resulting in a incomplete or corrupt file. Hence it is strongly recommended that one explicitly close a file after writing to a file, i.e.,

```
define write_image_to_file (file, img)
{
   variable fptr = fits_open_file ("new.fits", "c");
   fits_write_image (fptr, NULL, img, NULL, NULL);
   fits_write_date (fptr);
   fits_close_file (fptr);
}
```

2.3 Keywords

The high-level interface contains several functions for manipulating header keywords and records.

2.3.1 Reading Header Keywords

The `fits_read_key` may be used to read the values of one or more keywords. Suppose that the file `casA.fits` contains the following keywords in an extension called "EVENTS":

```
TELESCOP= 'CHANDRA ' / Telescope
INSTRUME= 'ACIS    ' / Instrument
DETNAM  = 'ACIS-7  ' / Detector
GRATING = 'NONE    ' / Grating
OBJECT  = 'CAS A   ' / Source name
RA_NOM  =     350.91781217089 / Nominal RA
DEC_NOM =     58.792819089577 / Nominal Dec
ROLL_NOM=     323.38710408328 / Nominal Roll
```

The `fits_read_key` function may be used to read, e.g, the OBJECT and RA_NOM keywords:

```
(obj, ra) = fits_read_key ("casA.fits[EVENTS]", "OBJECT", "RA_NOM");
```

After the function call, the variables `obj` and `ra` will have the data types `String_Type` and `Double_Type`, resp. If the requested keyword does not exist in the header, the function will return `NULL` to signal its not existence:

```
exptime = fits_read_key ("casA.fits[EVENTS]", "EXPTIME");
if (exptime == NULL)
  {
     message ("*** Warning: EXPTIME does not exist.  Assuming 3.2);
     exptime = 3.2;
  }
```

The `fits_read_key_struct` is an alternative to `fits_read_key` that returns a structure with field names that correspond to the keyword names. In most cases, a field name will just be the lower case version of the keyword name. However, if the keyword name does not start with an alphabetic character or contains a hyphen, then it will be normalized as follows:

1. The keyword name will be lowercased.
2. All non-alphanumeric characters will be changed to an underscore.
3. If the first character of the resulting name is numeric, then the name will be prefixed with an underscore.

To illustrate the normalization process, consider:

```
keys = fits_read_key_struct ("foo.fits", "OBJECT", "2CRVL3",
                              "DATE-OBS");
```

After the function call, `keys` will be a structure with the 3 fields: `object`, `_2crvl3`, and `date_obs`. If any of these keywords do not exist in the header, the value of the corresponding structure field will be NULL.

2.3.2 Writing Header Keywords

The `fits_update_key` function may be used to write or update the value of a keyword. If a keyword of the specified name exists, then the value of the keyword will be updated to the new value. Otherwise a new keyword will be appended to the header.

Other specialized keyword writing routines include `fits_write_date`, which write the current date in the required format, and `fits_write_chksum`, which computes and updates the checksum of the HDU. Finally the `fits_write_comment` and `fits_write_history` functions may be used to write comments and history records to the header, respectively.

2.4 Binary Tables

2.4.1 Reading Binary Tables

There are a several functions for reading binary tables. The simplest one, `fits_read_table` reads the entire binary table into a structure, whose fields correspond to the names of the columns in the table. (If a column has a name that contains non-alphanumeric characters, or does not start with an alphabetic character, then the structure field name for the column will be undergo the normalization process described for keywords.) For example, consider a file called `foo.fits` with a binary table whose structure is defined by the FITS header:

```
XTENSION= 'BINTABLE'
BITPIX  =                    8
NAXIS   =                    2
NAXIS1  =                   34
NAXIS2  =                  500
PCOUNT  =                    0
GCOUNT  =                    1
TFIELDS =                    4
TTYPE1  = 'TIME    '
TFORM1  = 'D       '
TTYPE2  = 'X       '
TFORM2  = 'E       '
TTYPE3  = 'Y       '
TFORM3  = 'E       '
TTYPE4  = 'PHAS    '
TFORM4  = '9I      '
EXTNAME = 'EXAMPLE '
TDIM4   = '(3,3)   '
```

This header shows that the binary table is 500 rows in length and contains 4 columns with names TIME, X, Y, and PHAS. The table may be read via

```
tbl = fits_read_table ("foo.fits[EXAMPLE]");
```

assigning a structure to the `tbl` variable. The structure has fields with names `time`, `x`, `y`, and `phas`, which be displayed via

```
vmessage ("tbl.time = %S", tbl.time);
vmessage ("tbl.x = %S", tbl.x);
vmessage ("tbl.y = %S", tbl.y);
vmessage ("tbl.phas = %S", tbl.y);
```

producing:

```
tbl.time = Double_Type[500]
tbl.x = Float_Type[500]
tbl.y = Float_Type[500]
tbl.z = Short_Type[500,3,3]
```

Note that the `fits_read_table` function not only read the data in a way that preserved the data type, but it also correctly identified the `phas` column as one containing a 3x3 image in every row!

Often one is interested in only a few columns of a table. Instead of reading the entire table, which could use a lot of memory for a large table, the `fits_read_table` function may also be used to read just the specified columns, e.g.,

```
 tbl = fits_read_table ("foo.fits[EXAMPLE]", "x", "y");
```

will read just the `X` and `Y` columns.

An alternative interface with much the same functionality is provided by the `fits_read_col` function. Instead of returning the data as a structure, it returns the data as multiple return values, e.g.,

```
t = fits_read_col ("foo.fits[EXAMPLE]", "time");
(x,y) = fits_read_col ("foo.fits[EXAMPLE]", "x", "y");
```

The `fits_read_cell` function may be used to read a single cell in the table. For example

```
phas = fits_read_cell ("foo.fits[EXAMPLE]", "phas", 4);
```

will return the 3x3 array of the "phas" column in the fourth row. Finally, the `fits_read_cells` function may be used to read a specified range of rows in the table. For instance,

```
(x,y) = fits_read_cells ("foo.fits[EXAMPLE]", "x", "y", 1, 1000);
```

will read the first 1000 rows of the `X` and `Y` columns in the table.

2.4.2 Writing Binary Tables

The high-level interface has several functions that are useful for the creation of a binary table. Chief among them is the `fits_write_binary_table` function, which supports several methods of calling it. The simplest use was illustrated earlier. Here more complicated uses will be considered.

As a first step, suppose that the binary table is to contain data for the Lissajous curve constructed as follows:

```
A_x = 10.0; omega_x = 3.0;  phi_x = 0.0;
A_y = 20.0; omega_y = 7.0;  phi_y = 1.0;
t = [0:100:0.01];
x = A_x * cos (omega_x*t + phi_x);
y = A_y * cos (omega_y*t + phi_y);
```

The goal is to write out arrays `t`, `x`, and `y` to a binary table called LISSAJOUS, and with columns of the corresponding names. The easiest way is to use:

```
data = struct {t, x, y}; data.t = t; data.x = x; data.y = y;
fits_write_binary_table ("foo.fits", "LISSAJOUS", data);
```

Now suppose that it is desired to write the parameters defining the Lissajous pattern as keywords and to write a history record to the file. One way to do this is via the `fits_update_key`, `fits_write_history`, and `fits_write_comment` functions:

```
fp = fits_open_file ("foo.fits[LISSAJOUS]");
fits_write_comment (fp, "This table contains data for a Lissajous pattern");
fits_update_key (fp, "A_X", A_x, "x(t) Amplitude");
fits_update_key (fp, "A_Y", A_y, "y(t) Amplitude");
fits_update_key (fp, "OMEGA_X", omega_x, "x(t) omega");
fits_update_key (fp, "OMEGA_Y", omega_y, "y(t) omega");
fits_update_key (fp, "PHI_X", phi_x, "x(t) phase");
fits_update_key (fp, "PHI_Y", phi_y, "y(t) phase");
fits_write_history (fp, "This was written as an example for the " +
                        "documentation of the slang cfitsio module");
fits_close_file (fp);
```

The advantage of using the `fits_update_key` is that it allows control over the comment associated with the keyword; however, repeated calls to `fits_update_key` can become tedious.

A simpler mechanism to achieve this goal is to pass the keywords and history information to the `fits_write_binary_table` function as an optional arguments,

```
keys = struct {A_x, omega_x, phi_x, A_y, omega_y, phi_y};
set_struct_fields (keys, A_x, omega_x, phi_x, A_y, omega_y, phi_y);
hist = struct {history, comment};
hist.comment = "This table contains data for a Lissajous pattern";
hist.history = "This was written as an example for the " +
                "documentation of the slang cfitsio module";
fits_write_binary_table ("foo.fits", "LISSAJOUS", data, keys, hist);
```

to produce:

```
XTENSION= 'BINTABLE' / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                   24 / width of table in bytes
NAXIS2  =                10000 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
```

```
TFIELDS =                    3 / number of fields in each row
TTYPE1  = 't       ' / label for field   1
TFORM1  = 'D       ' / data format of field: 8-byte DOUBLE
TTYPE2  = 'x       ' / label for field   2
TFORM2  = 'D       ' / data format of field: 8-byte DOUBLE
TTYPE3  = 'y       ' / label for field   3
TFORM3  = 'D       ' / data format of field: 8-byte DOUBLE
EXTNAME = 'LISSAJOUS' / name of this binary table extension
A_X     =                   10
OMEGA_X =                    3
PHI_X   =                    0
A_Y     =                   20
OMEGA_Y =                    7
PHI_Y   =                    1
COMMENT This table contains data for a Lissajous pattern
HISTORY This was written as an example for the documentation of the slang cfitsi
HISTORY o module
```

It is important to note that in the the above examples, the name of the file to contain the binary table was explicitly passed to the `fits_write_binary_table` function. This causes `fits_write_binary_table` to create a *new* file containing the binary table, and if a file of that name exists, *it will be deleted* before the new one is created.

To append a table to an existing file, first open it using the `fits_open_file` function, and then use the file pointer in place of the name:

```
fp = fits_open_file ("foo.fits", "w");   % <<-- note the "w"
   .
   .
fits_write_binary_table (fp, ....);
fits_close_file (fp);
```

This technique must also be used to create a file containing multiple binary tables:

```
fp = fits_open_file ("foo.fits", "c");   % <<-- note the "c"
   .
   .
fits_write_binary_table (fp, ....);      % first table
   .
   .
fits_write_binary_table (fp, ....);      % second table
fits_close_file (fp);
```

2.5 Images

2.5.1 Preliminaries

Dealing with FITS images in **S-lang** is easy as long as one understands that FITS stores images in FORTRAN *column-major* order, whereas **S-lang** utilizes a C *row-major* order. That is, the first dimension of a FITS array varies fastest whereas it is the last dimension of a **S-lang** array that varies fastest. This difference is automatically accounted for by the underlying `cfitsio` module. In other words, images may be used in **S-lang** scope as ordinary **S-lang** arrays where the first dimension varies slowest, and the `cfitsio` module will make the necessary translations when reading or writing an image from a file. An easy way to remember the **S-lang** or C ordering is that for a 2-d array, the first index is a row index and the second a column— the same as matrices are indexed in linear algebra. Do not fall into the trap of indexing a **S-lang** array the same as you would of indexing a point in Cartesian space (x,y), instead think in terms of rows and columns.

2.5.2 Reading and Writing Images

The `fits_read_img` may be used to read the image from the primary FITS HDU or a FITS image extension. It is not designed to read images that are stored in binary tables— there are other functions for that purpose. The `fits_read_img` function simply returns the data as an array of the appropriate type and dimensions (in row-major order) with any scaling defined via the BZERO and BSCALE header keywords applied.

Writing an image HDU is somewhat more involved than reading one because in addition to writing the image data, the header must first be set up to describe the image. Fortunately, the high-level functions make this easy.

Suppose that one has created an image array via, e.g., the `histogram` module's `hist2d` function from the X and Y columns of a binary table:

```
(x,y) = fits_read_col ("evt2.fits[EVENTS]", "X","Y");
xgrid = 3840.5 + [0:1023:2];
ygrid = 3840.5 + [0:1023:2];
img = hist2d (y, x, ygrid, xgrid);
```

and that one wants to write this out to a fits file called `img.fits`. The simplest way to do this is using `fits_write_image_hdu`:

```
fits_write_image_hdu ("img.fits", NULL, img);
```

Note that the data-type of the image array controls the type of image written to the fits file. If the image array is an array of `Double_Types`, then the image will be written with BITPIX set to -64. To have such an array out as 16 bit integers, then the array must first be scaled to the range of a 16 bit integer and then typecast to `Int16_Type`:

```
int16_image = typecast (img, Int16_Type);
```

Additional keywords may be written to the image HDU using the `fits_update_key` function. And like `fits_write_binary_table`, the `fits_write_image_hdu` function takes optional parameters that specify additional keywords, history, and comments to be written. The reader is referred to the discussion of the `fits_write_binary_table` function for more information.

2.6 WCS Routines

2.6.1 Introduction

The FITS package includes a set of routines for reading and writing *WCS* `http://fits.gsfc.nasa.gov/fits_wcs.html` keywords in the form proposed by

Greisen and Calabretta `http://www.atnf.csiro.au/people/mcalabre/WCS/wcs.pdf` . Although they are part of the high-level interface, the routines are somewhat experimental and as such must be loaded separately via:

```
require ("fitswcs");
```

The routines in this interface deal with a structure that describes the WCS via the following fields:

naxis

The number of axes to transform (Int_Type)

ctype

Specifies the WCS transformation (String_Type[naxis])

cunit

Units (String_Type[naxis])

crval

WCS values at the reference pixel (Double_Type[naxis])

crpix

The coordinates of the reference pixel (Double_Type[naxis])

cdelt

Species the gradient at the reference pixel (Double_Type[naxis])

pc

An array used to linearly transform the WCS. (Double_Type[naxis,naxis] or NULL)

pv

An array of addition parameters used to specify the WCS (NULL in most cases)

ps

An array of additional string parameters (NULL in most cases).

wcsname

A name given to this coordinate system.

While the user is encouraged to understand the FITS WCS conventions and the precise meanings of these fields, the `fitswcs` interface provides routines to make the use of this structure as transparent as possible for the most common uses. A few examples will illustrate this.

2.6.2 Examples

Consider once again the example of creating a FITS image by binning two columns of a FITS binary table:

```
(x,y) = fits_read_col ("evt2.fits[EVENTS]", "X","Y");
xgrid = 3840.5 + [0:1023:2];
ygrid = 3840.5 + [0:1023:2];
img = hist2d (y, x, ygrid, xgrid);
fits_write_image_hdu ("img.fits", NULL, img);
```

Unfortunately the resulting file will contain none of the WCS information that was attached to the X and Y columns from which the image was constructed. One might be tempted to simply copy that information to the output file with the aid of the `fitswcs` routines via

```
wcs = fitswcs_get_column_wcs ("evt2.fits[EVENTS]", ["Y", "X"]);
fitswcs_put_img_wcs ("img.fits", wcs);
```

The problem with this approach is that the WCS read from the binary table does not describe the image created from it because it knows nothing about how the image was binned nor how the image pixel coordinates relate back to the X and Y columns. That information is contained in the definition of the grids passed to the `hist2d` function:

```
xgrid = 3840.5 + [0:1023:2];
ygrid = 3840.5 + [0:1023:2];
```

These grids describe a simple linear transformation from image pixel coordinates to the (X,Y) coordinates of the binary table. Since the transformation is linear, the `fitswcs_bin_wcs` function may be used to transform the WCS:

```
wcs = fitswcs_bin_wcs (wcs, ygrid, xgrid);
```

It is the transformed WCS that is to be written out:

```
fitswcs_put_img_wcs ("img.fits", wcs);
```

It is important to note the order in which the X and Y arguments were used. Recall that FITS stores images in a FORTRAN column-major order whereas **S-lang** uses a row-major order. For this reason, "row-like" parameters come before "column-like" parameters in statements such as

```
img = hist2d (y, x, ygrid, xgrid);
wcs = fitswcs_get_column_wcs ("evt2.fits[EVENTS]", ["Y", "X"]);
wcs = fitswcs_bin_wcs (wcs, ygrid, xgrid);
```

2.6.3 Alternate WCS

Sometimes it is useful to attach more than one coordinate system to an image. For example, it is useful to have a coordinate system that maps the pixel coordinates back to (X,Y) coordinates from which they were derived. The `fitswcs_new_img_wcs` may be used to construct a linear WCS corresponding to the linear coordinate grids of the image:

```
wcsP = fitswcs_new_img_wcs (ygrid, xgrid);
wcsP.wcsname = "PHYSICAL";
fitswcs_put_img_wcs ("img.fits", wcsP, 'P');
```

Note that the WCS was given the name “PHYSICAL”. While not required, this enables this alternate coordinate system to be displayed as the physical system by the DS9 image display program.

2.6.4 Degenerate Axes

Consider a FITS file `hydra.fits` containing an image HDU with the following FITS header:

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                  -32 / number of bits per data pixel
NAXIS   =                    4 / number of data axes
NAXIS1  =                  657 / length of data axis
NAXIS2  =                  657 / length of data axis
NAXIS3  =                    1 / length of data axis
NAXIS4  =                    1 / length of data axis
CTYPE1  = 'RA---SIN'
CRVAL1  =          139.5235701
CRPIX1  =                  330
CDELT1  =    -0.0004166666768
CTYPE2  = 'DEC--SIN'
CRVAL2  =       -12.0955450949
CRPIX2  =                  328
CDELT2  =     0.0004166666768
PC1_1   =                    1
PC1_2   =-1.7318465835227e-09
PC2_1   = 1.7318465835227e-09
PC2_2   =                    1
CTYPE3  = 'FREQ    '
CRVAL3  =         332902343.75
CRPIX3  =                    1
CDELT3  =            2490234.5
CTYPE4  = 'STOKES  '
CRVAL4  =                    1
CRPIX4  =                    1
CDELT4  =                    1
```

This particular image had so-called “degenerate axes” added, which had the effect of increasing its dimensionality from 2 to 4. As such, this image may be rejected by some image display programs that expect a 2-d image. In fact,

```
img = fits_read_img ("hydra.fits");
wcs = fitswcs_get_img_wcs ("hydra.fits");
```

will read the image as a `Float_Type[1,1,657,657]` object, and the WCS as a 4-d with wcs.ctype equal to

```
["STOKES", "FREQ", "DEC--SIN", "RA---SIN"]
```

The degenerate dimensions may be removed from the image via

```
img = img[0,0,*,*];
```

producing a 2d image of type `Float_Type[657,657]`. The corresponding wcs may be obtained using the `fitswcs_slice` function to extract the last two dimensions of the WCS:

```
wcs = fitswcs_slice (wcs, [2,3]);
```

Another use of the `fitswcs_slice` is to reorder the dimensions of the WCS. For example, earlier it was pointed out that when constructing an image from columns in a table, that one read the WCS in a row-major order. If the reverse order was used when obtaining the WCS from the columns of a binary table, e.g.,

```
wcs = fitswcs_get_column_wcs ("evt2.fits[EVENTS]", ["X", "Y"]);
```

then it would have been necessary to reverse the order of the dimensions of the WCS structure. The `fitswcs_slice` may be used to swap the dimensions of the WCS, e.g.,

```
wcs = fitswcs_slice (wcs, [1,0]);
```

2.7 High-level Function Reference

2.7.1 fits_read_errmsgs

Synopsis

Retrieve all error messages from the CFITSIO error stack

Usage

`String_Type[] fits_read_errmsgs ()`

Description

This function returns all the error messages from the CFITSIO error message stack as an array of strings.

Notes

Using this function will cause the error message stack to be cleared.

See Also

`3.2.2 (_fits_read_errmsg), 2.7.2 (fits_set_verbose_errors)`

2.7.2 fits_set_verbose_errors

Synopsis

Set the verbosity level of the cfitsio error messages

Usage

`fits_set_verbose_errors (Int_Type level)`

Description

When a call to a function in the high-level interface fails, a error message will get generated. By default, all messages from the underlying cfitsio error stack are printed. This behavior may be turned off by calling this function with `level` equal to 0.

See Also

2.7.1 (fits_read_errmsgs)

2.7.3 fits_open_file

Synopsis

Open a fits file

Usage

`Fits_File_Type fits_open_file (String_Type filename, String_Type mode)`

Description

The `fits_open_file` function can be used to open and existing fits file for reading or updating, or to create a new fits file, depending upon the value of the `mode` parameter. Specifically, if `mode` is `"r"`, the file will be opened for reading. If `mode` is `"w"`, the file will be opened for updating (both reading and writing). Otherwise, `mode` must be `"c"`, which indicates that a new file is to be created. In the latter case, if a file already exists with the specified name, it will get deleted and a new one created in its place.

If the function fails, it will signal an error; otherwise an open file pointer will be returned.

See Also

2.7.4 (fits_close_file), 2.7.17 (fits_create_binary_table)

2.7.4 fits_close_file

Synopsis

Close a fits file

Usage

`fits_close_file (Fits_File_Type f)`

Description

The `fits_close_file` closes a previously opened fits file. The function will signal an error if the operation fails.

Notes

This function could fail if it fails to write out any buffered data because of filesystem errors (disk full, etc.).

See Also

`2.7.3 (fits_open_file)`

2.7.5 fits_move_to_interesting_hdu

Synopsis

Move to an extension that looks interesting

Usage

```
fits_move_to_interesting_hdu (fp [, hdu_type]

      Fits_File_Type fp;
      Int_Type hdu_type;
```

Description

The function move the fits file pointer `fp` forward to an HDU that looks interesting. By definition, an interesting HDU is one in which NAXIS is non-zero. The first parameter `fp` must be a pointer to an already open fits file. The second parameter, if present, may be used to specifiy the type of HDU, e.g., either an image (`hdu_type=_FITS_IMAGE_HDU`) or a binary table (`hdu_type=_FITS_BINARY_TBL`).

If the function fails to find an interesting HDU of the appropriate type, an exception will be generated.

See Also

`2.7.3 (fits_open_file)`

2.7.6 fits_key_exists

Synopsis

Check for the existence of a keyword

Usage

```
Int_Type fits_key_exists (fd, key)

      Fits_File_Type or String_Type fd;
      String_Type key;
```

Description

The `fits_key_exists` function checks for the existence of a specified keyword in the file specified by the descriptor `fd`, which must specify the name of a file or an open file pointer.

If the specified key exists, the function return `1`, otherwise it returns `0`.

See Also

`2.7.15 (fits_read_key), 2.7.13 (fits_read_header)`

2.7.7 fits_get_colnum

Synopsis

Get the column numbers of specified columns

Usage

```
column_num = fits_get_colnum (fd, column_name)

       Fits_File_Type or String_Type fd;
       String_Type column_name;
```

Description

This function returns the column number of the column with the specified name. The file-descriptor `fd` must specify the name of a file, or an open fits file pointer.

See Also

`2.7.8 (fits_binary_table_column_exists)`

2.7.8 fits_binary_table_column_exists

Synopsis

Check for the existence of a binary table column

Usage

```
Int_Type fits_binary_table_column_exists (fd, col)

       Fits_File_Type or String_Type fd;
       String_Type col;
```

Description

This function may be used to determine whether or not a named column exists in a binary table. The table is specified via the `fd` parameter which must either be the name of a file containing the binary table, or an file pointer.

If the specified column exists, `1` will be returned; otherwise the function will return `0`.

See Also

`2.7.6 (fits_key_exists), 2.7.3 (fits_open_file)`

2.7.9 fits_read_col

Synopsis

Read one or more columns from a FITS binary table

Usage

```
(x1, ...xN) = fits_read_col (file, c1, ...  cN)

       Fits_File_Type or String_Type file;
       Int_Type or String_Type c1, ...cN;
```

Description

This function returns one or more vectors containing objects in the specified columns of the binary table indicated by `file`. If `file` is a string, then the file will be opened via the virtual file specification implied by `file`. Otherwise, `file` should represent an already opened FITS file. The column parameters may either be strings denoting the column names, or integers representing the column numbers.

See Also

`2.7.11 (fits_read_cell), 2.7.12 (fits_read_row), 2.7.14 (fits_read_table)`

2.7.10 fits_read_col_struct

Synopsis

Read one or more columns from a FITS binary table

Usage

```
struct = fits_read_col_struct (file, col1, ...)

        Fits_File_Type or String_Type file;
        String_Type col1, ...;
```

Description

This function works exactly like `fits_read_col` except it returns the values in a structure. See the documentation on that function for more information.

See Also

`2.7.9 (fits_read_col), 2.7.16 (fits_read_key_struct), 2.7.12 (fits_read_row), 2.7.13 (fits_read_header)`

2.7.11 fits_read_cell

Synopsis

Read a cell from a FITS binary table

Usage

```
X = fits_read_cell (file, c, r)

       Fits_File_Type or String_Type file;
       Int_Type r, c;
```

Description

This function returns the object in the column `c` and row `r` of the binary table indicated by `file`. If `file` is a string, then the file will be opened via the virtual file specification implied by `file`. Otherwise, `file` should represent an already opened FITS file.

See Also

`2.7.9 (fits_read_col), 2.7.12 (fits_read_row)`

2.7.12 fits_read_row

Synopsis

Read a row from a FITS binary table

Usage

```
Struct_Type fits_read_cell (file, r)

       Fits_File_Type or String_Type file;
       Int_Type r;
```

Description

This function returns a structure containing the data in the columns of the row `r` of the binary table indicated by `file`. If `file` is a string, then the file will be opened via the virtual file specification implied by `file`. Otherwise, `file` should represent an already opened FITS file.

See Also

2.7.9 (fits_read_col), 2.7.11 (fits_read_cell)

2.7.13 fits_read_header

Synopsis

Read a FITS header

Usage

```
Struct_Type fits_read_header (file)

       Fits_File_Type or String_Type file;
```

Description

This function reads the header of the fits file given by the `file` parameter and returns it as a structure. If `file` is a string, then the file will be opened via the virtual file specification implied by `file`. Otherwise, `file` should represent an already opened FITS file.

See Also

2.7.14 (fits_read_table)

2.7.14 fits_read_table

Synopsis

Read a FITS table

Usage

```
Struct_Type fits_read_table (file [,columns...])

       Fits_File_Type or String_Type file;
```

Description

`fits_read_table` reads the data in a table of the FITS file specified by `file` and returns it as a structure. If the optional column name parameters are specified, then only those columns will be read. Otherwise, the entire table will be returned.

If `file` is a string, then the file will be opened via the virtual file specification implied by `file`. Otherwise, `file` should represent an already opened FITS file.

See Also

`2.7.9 (fits_read_col), 2.7.11 (fits_read_cell), 2.7.12 (fits_read_row), 2.7.13 (fits_read_header)`

2.7.15 fits_read_key

Synopsis

Read one or more keywords from a FITS file

Usage

```
(val1,...)  = fits_read_key (file, key1, ...)

        Fits_File_Type or String_Type file;
        String_Type key1, ...;
```

Description

`fits_read_key` reads the values of one or more keywords in the fits file specified by `file` and returns them. If `file` is a string, then the file will be opened via the virtual file specification implied by `file`. Otherwise, `file` should represent an already opened FITS file. If any of the keywords do not exist, a value of `NULL` will be returned for the corresponding keyword.

See Also

`2.7.16 (fits_read_key_struct), 2.7.9 (fits_read_col), 2.7.11 (fits_read_cell), 2.7.12 (fits_read_row), 2.7.13 (fits_read_header)`

2.7.16 fits_read_key_struct

Synopsis

Read one or more keywords from a FITS file

Usage

```
struct = fits_read_key (file, key1, ...)

        Fits_File_Type or String_Type file;
        String_Type key1, ...;
```

Description

This function works exactly like `fits_read_key` excepts returns the values in a structure. See the documentation on that function for more information.

See Also

2.7.15 (fits_read_key), 2.7.9 (fits_read_col), 2.7.11 (fits_read_cell),
2.7.12 (fits_read_row), 2.7.13 (fits_read_header)

2.7.17 fits_create_binary_table

Synopsis

Prepare a binary table

Usage

```
fits_create_binary_table (file, extname, nrows, ttype, tform, tunit)
```

```
Fits_File_Type or String_Type file;
String_Type extname;
Int_Type nrows;
String_Type ttype[];
String_Type tform[];
String_Type tunit[];
```

Description

This creates a new binary table with the specified structure. The parameters `ttype`, `tform`, and `tunit` are string arrays that specify the column names, column data type, and column units, respectively. The binary table will be given the extension name `extname`.

See Also

2.7.18 (fits_write_binary_table), 2.7.3 (fits_open_file)

2.7.18 fits_write_binary_table

Synopsis

Write a binary table

Usage

```
fits_write_binary_table (file, extname, sdata, [skeys [,hist]])
```

```
Fits_File_Type or String_Type file;
String_Type extname;
Struct_Type sdata;
Struct_Type skeys;
Struct_Type hist;
```

Description

The `fits_write_binary_table` function creates a new binary table in the specified file. The parameter `file` specifies either a filename or an open file pointer. The `extname` parameter specifies the extension name of the binary table. The data written to table are specified in the `sdata` structure, where the name of the structure field specifies the column name. If `skeys` is non-NULL, then it is a structure indicating additional keywords to be written to the header of the binary table. If the optional parameter `hist` is present and non-NULL, then it is a structure whose fields indicate either comment or history information to be written to the header.

Example

The following code

```
variable data = struct { x, cosx, sinx };
data.x = [0:2*PI:0.01];
data.cosx = cos(data.x);
data.sinx = sin(data.x);

variable keys = struct { hduname, username};
keys.hduname = "COSXSINX";
keys.username = "John Doe";

variable hist = struct { history, comment};
hist.history = ["This is a history record", "This is another"];
hist.comment = ["This is a comment", "And this is another"];

fits_write_binary_table ("foo.fits", "COSXSINX", data, keys, hist);
```

produces a binary table with the header:

```
XTENSION= 'BINTABLE' / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                   24 / width of table in bytes
NAXIS2  =                  629 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                    3 / number of fields in each row
TTYPE1  = 'x       ' / label for field   1
TFORM1  = 'D       ' / data format of field: 8-byte DOUBLE
TTYPE2  = 'cosx    ' / label for field   2
TFORM2  = 'D       ' / data format of field: 8-byte DOUBLE
TTYPE3  = 'sinx    ' / label for field   3
TFORM3  = 'D       ' / data format of field: 8-byte DOUBLE
EXTNAME = 'COSXSINX' / name of this binary table extension
HDUNAME = 'COSXSINX'
USERNAME= 'John Doe'
HISTORY This is a history record
HISTORY This is another
COMMENT This is a comment
COMMENT And this is another
```

Notes

This function provides no mechanism to mix comments and keyword records. As the example shows, this function places the comment and history records at the end of the table.

See Also

`2.7.17 (fits_create_binary_table), 2.7.3 (fits_open_file)`

2.7.19 fits_update_key

Synopsis

Update the value of a keyword

Usage

```
fits_update_key (fd, key, val [,comment])
```

```
String_Type or Fits_File_Type fd;
String_Type key;
Any type val;
String_Type comment;
```

Description

The `fits_update_key` function updates the value and comment fields of an existing keyword with the specified name. If the keyword does not exist, a new keyword will be appended to the end of the header.

See Also

2.7.20 (fits_update_logical), 2.7.15 (fits_read_key)

2.7.20 fits_update_logical

Synopsis

Update the value of a logical (boolean) keyword

Usage

```
fits_update_logical (fd, key, val, comment)
```

```
String_Type or Fits_File_Type fd;
String_Type key;
Any type val;
String_Type comment;
```

Description

The `fits_update_logical` function updates the value and comment fields of an existing keyword of the specified name with the specified boolean value. If the keyword does not exist, a new keyword will be appended to the end of the header.

See Also

2.7.19 (fits_update_key)

2.7.21 fits_write_comment

Synopsis

Write a comment to the header

Usage

```
fits_write_comment (fd, comment)
```

```
Fits_File_Type or String_Type fd;
String_Type comment;
```

Description

As the name indicates, this function writes a comment record to the specified fits file. The file-descriptor `fd` must either be the name of a fits file or an open fits file pointer.

See Also

`2.7.19 (fits_update_key), 2.7.22 (fits_write_history)`

2.7.22 fits_write_history

Synopsis

Write a history record to the header

Usage

`fits_write_history (fd, history)`

```
Fits_File_Type or String_Type fd;
String_Type history;
```

Description

As the name indicates, this function writes a history record to the specified fits file. The file-descriptor `fd` must either be the name of a fits file or an open fits file pointer.

See Also

`2.7.19 (fits_update_key), 2.7.21 (fits_write_comment)`

2.7.23 fits_write_date

Synopsis

Write the DATE keyword to the current HDU

Usage

`fits_write_date (fd)`

```
Fits_File_Type or String_Type fd;
```

Description

The `fits_write_date` function calls `_fits_write_date` to write the DATE to the header of the specified file descriptor, which must either be the name of a fits file or an open fits file pointer.

See Also

`2.7.19 (fits_update_key)`

2.7.24 fits_write_chksum

Synopsis

Compute and write the DATASUM and CHECKSUM keywords

Usage

```
fits_write_chksum (fd)

       Fits_File_Type or String_Type fd;
```

Description

The `fits_write_chksum` function calls `_fits_write_comment` to compute and write the DATASUM and CHECKSUM keywords to the header of the specified file descriptor, which must either be the name of a fits file or an open fits file pointer.

See Also

2.7.19 (fits_update_key), 2.7.25 (fits_verify_chksum)

2.7.25 fits_verify_chksum

Synopsis

Verify the checksums for the current HDU

Usage

```
isok = fits_verify_chksum (fd [,dataok, hduok])

       Fits_File_Type or String_Type fd;
       Ref_Type dataok, hduok;
```

Description

The `fits_verify_chksum` function calls `_fits_verify_chksum` to verify the header and data checksums of the current HDU. A non-zero return value signifies that the checksums are ok, otherwise the function returns 0 to indicate that the checksums are invalid. The individual checksums of the HDU or data can be checked through the use of the optional parameters.

See Also

2.7.24 (fits_write_chksum)

2.7.26 fits_read_records

Synopsis

Read all the records in a fits header

Usage

```
String_Type[] fits_read_records (Fits_File_Type or String_Type fp)
```

Description

This function returns a list of all the header records associated with the fits file descriptor as an array of strings.

See Also

2.7.27 (fits_write_records), 2.7.15 (fits_read_key)

2.7.27 fits_write_records

Synopsis

Write records to fits header

Usage

`fits_write_records (fd, records)`

```
Fits_File_Type or String_Type fd;
Array_Type records;
```

Description

This function uses the `_fits_write_record` function to write a series of records to the current HDU.

See Also

2.7.26 (fits_read_records)

2.7.28 fits_get_keyclass

Synopsis

Obtain the key classes for a set of cards

Usage

`Int_Type[] = fits_get_keyclass (Array_Type cards)`

Description

This function uses the `_fits_get_keyclass` function to obtain the key-classes associated with one or more cards. The function returns an integer-valued array of the same length as the `cards` array.

Example

Obtain set of header cards to those that are not associated with the cards describing the structure of the HDU:

```
variable cards = fits_read_records ("evt2.fits[EVENTS]");
variable classes = fits_get_keyclass (cards);
cards = cards[where (classes != _FITS_TYP_STRUC_KEY)];
```

See Also

2.7.26 (fits_read_records), 2.7.15 (fits_read_key)

2.7.29 fits_get_bitpix

Synopsis

Get the fits bitpix value for an array

Usage

```
Int_Type fits_get_bitpix (array)
```

Description

This function may be used to obtain the bitpix value for a specified image array. The array must be an integer or floating point type, otherwise and error will be generated. The bitpix value is returned.

See Also

`2.7.32 (fits_write_image_hdu), 2.7.30 (fits_read_img)`

2.7.30 fits_read_img

Synopsis

Read image data from a fits file

Usage

```
Array_Type fits_read_img (fd)

       Fits_File_Type or String_Type fd;
```

Description

This function reads an image from the specified file descriptor. The file descriptor must be either the name of an existing file, or an open file pointer. It returns the image upon sucess, or signals an error upon failure.

See Also

`2.7.14 (fits_read_table), 2.7.9 (fits_read_col), 2.7.3 (fits_open_file),`
`2.7.33 (fits_write_img)`

2.7.31 fits_create_image_hdu

Synopsis

Create a primary array or image extension

Usage

```
fits_create_image_hdu (fd, extname, type, dims)

       Fits_File_Type or String_Type fd;
       String_Type extname;
       Array_Type dims;
       DataType_Type type;
```

Description

This function make use of the `_fits_create_img` function to create an image extension or primary array of the specified type and size. If the `extname` parameter is non-NULL, then an EXTNAME keyword will be written out with the value of the extname parameter. The `dims` parameter must be a 1-d integer array that corresponds to the dimensions of the array to be written.

If `fd` is specified as a string, then a new file of that name will be created. If a file by that name already exists, it will be deleted and a new one created. If this behavior is undesired, then explicitly open the file and pass this routine the resulting file pointer.

See Also

2.7.32 (fits_write_image_hdu)

2.7.32 fits_write_image_hdu

Synopsis

Write an image extension

Usage

```
fits_write_image_hdu (file, extname, image [,skeys [,hist]])
```

```
Fits_File_Type or String_Type file;
String_Type extname;
Any_Type image
Struct_Type skeys;
Struct_Type hist;
```

Description

The `fits_write_image_hdu` function creates a new image extension in the specified file. The parameter `file` specifies either a filename or an open file pointer. The `extname` parameter specifies the extension name of the image, or NULL for the primary image. The image data written to the file are specified by the `image` parameter. If the optional parameter `skeys` is non-NULL, then it is a structure indicating additional keywords to be written to the HDU. If the optional parameter `hist` is present and non-NULL, then it is a structure whose fields indicate either comment or history information to be written to the header.

Example

The following code

```
variable img = [1:128*128]; reshape (img, [128,128]);
variable keys = struct { hduname, username};
keys.hduname = "MY_IMAGE";
keys.username = "John Doe";
variable hist = struct { history, comment};
hist.history = ["This is a history record", "This is another"];
hist.comment = ["This is a comment", "And this is another"];
fits_write_image_hdu ("foo.fits", NULL, img, keys, hist);
```

produces an image HDU with the header:

```
SIMPLE  =                    T / file does conform to FITS standard
BITPIX  =                   32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                  128 / length of data axis 1
NAXIS2  =                  128 / length of data axis 2
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format is defined in 'Astronomy
COMMENT   and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H
HDUNAME = 'MY_IMAGE'
USERNAME= 'John Doe'
HISTORY This is a history record
HISTORY This is another
COMMENT This is a comment
COMMENT And this is another
```

Notes

This function provides no mechanism to mix comments and keyword records. As the example shows, this function places the comment and history records at the end of the table.

See Also

`2.7.17 (fits_create_binary_table), 2.7.3 (fits_open_file)`

2.7.33 fits_write_img

Synopsis

Write the image data to an Image HDU

Usage

```
fits_write_img (Fits_File_Type fptr, Any_Type data)
```

Description

This function writes the image data out to current HDU, assumed to be an Image HDU.

See Also

`2.7.32 (fits_write_image_hdu), 2.7.31 (fits_create_image_hdu)`

2.8 WCS Function Reference

2.8.1 fitswcs_new

Synopsis

Create a new-ndimensional linear WCS

Usage

```
wcs = fitswcs_new (Int_Type naxis)
```

Description

This function returns a new WCS structure of the specified dimensionality that represents an identity (linear) transformation.

See Also

`2.8.3 (fitswcs_get_img_wcs), 2.8.4 (fitswcs_get_column_wcs), 2.8.5 (fitswcs_get_vector_wcs)`

2.8.2 fitswcs_slice

Synopsis

Form a new wcs from one or more axes of another

Usage

`new_wcs = fitswcs_slice (wcs, dims)`

Description

This function may be used to construct a new wcs from another by rearranging its axes or by using a subset of them. The `dims` argument specifies the dimensions to use.

Example

Suppose that `wcs` represents a 4 dimensional WCS. Then

```
wcs2 = fitswcs_slice (wcs, [0,1]);
```

will result in a 2 dimensional WCS from the first 2 axis of the input WCS. Similarly,

```
wcs2 = fitswcs_slice (wcs, [1,0]);
```

will produce a 2d WCS with the first two axes swapped.

See Also

`2.8.3 (fitswcs_get_img_wcs), 2.8.4 (fitswcs_get_column_wcs), 2.8.5 (fitswcs_get_vector_wcs)`

2.8.3 fitswcs_get_img_wcs

Synopsis

Read a WCS for a FITS image

Usage

`wcs = fitswcs_get_img_wcs (fp [,alt])`

Description

The `fitswcs_get_img_wcs` returns a structure representing a WCS from the specified file descriptor `fp` corresponding to an image HDU. An optional parameter may be used to specified an alternate WCS.

Example

```
wcs = fitswcs_get_img_wcs ("img.fits[IMAGE]", 'P');
```

See Also

2.8.7 (fitswcs_put_img_wcs), 2.8.4 (fitswcs_get_column_wcs), 2.8.5 (fitswcs_get_vector_wcs)

2.8.4 fitswcs_get_column_wcs

Synopsis

Get the WCS attached to one or more columns of a binary table

Usage

`fitswcs_get_column_wcs (fp, columns-array [,alt]`

Description

This function may be used to obtain the WCS associated with one or more columns of a binary table. The file descriptor `fp` must specify a binary table. The `columns-array` parameter should be an array of columns names. The third parameter is optional and is used to specify an alternate WCS.

Example

```
wcs = fitswcs_get_column_wcs ("evt1.fits[EVENTS]", ["X","Y"]);
```

See Also

2.8.8 (fitswcs_put_column_wcs), 2.8.3 (fitswcs_get_img_wcs), 2.8.5 (fitswcs_get_vector_wcs)

2.8.5 fitswcs_get_vector_wcs

Synopsis

Get the WCS of an image in a specified table cell

Usage

`wcs = fitswcs_get_vector_wcs (fp, column_name, row [,alt])`

Description

This function reads the WCS of an image in a specified cell of a binary table given by `fp` parameter. The second and third parameters specify the column name and row number of the cell. An optional fourth parameter may be used to obtain the corresponding alternate WCS.

Example

This example reads the WCS associated with the image in the second row of the QEU column of the binary table with HDUNAME equal to AXAF_QEU1 in the file "HRCQEU.fits":

```
wcs = fitswcs_get_vector_wcs ("HRCQEU.fits[AXAF_QEU1], "QEU", 2);
```

Notes

The current implementation does not yet support references to the WCS of other cells.

See Also

`2.8.4 (fitswcs_get_column_wcs), 2.8.3 (fitswcs_get_img_wcs)`

2.8.6 fitswcs_new_img_wcs

Synopsis

Create a linear WCS for an image

Usage

`wcs = fitswcs_new_img_wcs (grid0,grid1,...)`

Description

This function may be used to construct a linear WCS for an image with the specified grids. The grids are assumed to be linear.

Example

Use the histogram module's hist2d function to create an image from the X and Y columns in a file, and the construct a corresponding WCS:

```
(x,y) = fits_read_col ("table.fits", "X", "Y");
gridx = [min(x):max(x):0.5];
gridy = [min(y):max(y):0.5];
img = hist2d (y,x,gridy,gridx);
wcs = fitswcs_new_img_wcs (gridy, gridx);
```

See Also

`2.8.1 (fitswcs_new), 2.8.3 (fitswcs_get_img_wcs)`

2.8.7 fitswcs_put_img_wcs

Synopsis

Write a WCS out to an image header

Usage

`fitswcs_put_img_wcs (fp, wcs [,alt])`

Description

The `fitswcs_put_img_wcs` may be used to write the specified wcs out to the image HDU specified by the `fp` parameter. An optional third parameter may be used to specify an alternate WCS.

Example

```
fp = fits_open_file ("img.fits", "w");
  .
  .
  .
fits_put_img_wcs (fp, wcs, 'P');
fits_close_file (fp);
```

See Also

2.8.8 (fitswcs_put_column_wcs)

2.8.8 fitswcs_put_column_wcs

Synopsis

Write the WCS attached to one or more table columns

Usage

```
fitswcs_put_column_wcs (fp, wcs, columns-array [,alt])
```

Description

This function may be used to attach a WCS to one or more columns of a binary table. The dimensionality of the specified WCS must match the length of the array specifying the column names. The first parameter, `fp` must specify a binary table extension. The fourth parameter is optional and may be used to specify an alternate WCS.

Example

```
fitswcs_put_column_wcs ("evt2.fits[EVENTS], wcs, ["X","Y"]);
```

See Also

2.8.4 (fitswcs_get_column_wcs), 2.8.7 (fitswcs_put_img_wcs), 2.8.3 (fitswcs_get_img_wcs)

2.8.9 fitswcs_linear_transform_wcs

Synopsis

Apply a linear transformation to a WCS

Usage

```
wcs1 = fitswcs_linear_transform_wcs (wcs, U0, A, X0)
```

```
    wcs: The specified WCS to transform
  U0,X0: 1-d arrays
      A: 2-d array (or 1-d array representing a diagonal matrix)
```

Description

This function may be used to create a new WCS by applying a linear transformation to an existing one. This is useful when one has a WCS associated with physical coordinates `X`, and then applies the linear transformation

```
U = U0 + A#(X-X0)
```

to the coordinates X. Then corresponding WCS for the resulting image is given by

```
new_wcs = fitswcs_linear_transform_wcs (wcs, U0, A, X0);
```

Notes

The dimensionality of the WCS is limited to 2 in the current implementation.

See Also

`2.8.10 (fitswcs_rebin_wcs), 2.8.11 (fitswcs_bin_wcs)`

2.8.10 fitswcs_rebin_wcs

Synopsis

This function may be used to obtain the wcs for a rebinned image

Usage

`wcs1 = fitswcs_rebin_wcs (wcs, old_dims, new_dims...)`

Description

This function may be used to construct the WCS for a rebinned image from the WCS of of the unbinned image. The grid parameters specify the linear grids the new image.

Example

new_img = hist2d_rebin (new_yrid, new_xgrid, old_ygrid, old_xgrid, old_img); new_wcs = fitswcs_rebin_wcs (old_wcs, array_shape(old_img), array_shape(new_img));

See Also

`2.8.11 (fitswcs_bin_wcs), 2.8.9 (fitswcs_linear_transform_wcs), 2.8.2 (fitswcs_slice)`

2.8.11 fitswcs_bin_wcs

Synopsis

This function may be used to obtain the wcs for a rebinned image

Usage

`wcs1 = fitswcs_rebin_wcs (wcs, grid0, grid1, ...)`

Description

This function may be used to construct the WCS for an image created by binning a set of coordinates from, e.g., a pixel-list. The `wcs` parameter represents the wcs attached to the unbinnned coordinates. The grid parameters specify the linear grids that were used to create the image.

See Also

`2.8.10 (fitswcs_rebin_wcs), 2.8.9 (fitswcs_linear_transform_wcs), 2.8.2 (fitswcs_slice)`

Chapter 3

The low-level interface

3.1 Overview

Functions in the low-level module are usually needed when it is necessary to perform some task that is not readily achievable using the high-level interface. This module may be loaded using

```
require ("cfitsio");
```

When mixing functions from both interfaces, it is not necessary to explicitly load the `cfitsio` module in this manner since it is loaded automatically by the high-level interface.

For the most part, for those functions that have been wrapped, the `cfitsio` module represents a 1-1 mapping between the functions of the cfitsio library and those of the module. For this reason a detailed description of the functions in the `cfitsio` module will not be given here; the reader is referred to the documentation for the CFITSIO library itself for the details. Here only the semantic differences between the functions in the module and those of the library, and how the functions are documented.

Most CFITSIO functions adhere to a so-called "inherited status" convention via a "status" argument as the last parameter. In addition functions also return the error status as a return value. For simplicity the wrapping by the module does not respect this convention. That is, none of the module's functions take a status argument. For example, the CFITSIO documentation for the `fits_get_num_hdus` specifies that it is to be called from C via:

```
status = fits_get_num_hdus (fptr, &hdunum, &status);
```

This function has been wrapped such that it is to be called from **S-lang** via

```
status = _fits_get_num_hdus (fptr, &hdunum);
```

3.2 Low-level Function Reference

3.2.1 _fits_get_errstatus

Synopsis

Retrieve a text string corresponding to an error code

Usage

`String_Type _fits_get_errstatus (Int_Type status)`

Description

This function is a wrapper around the cfitsio library function `fits_get_errstatus`. See its documentation for additional information.

See Also

`2.7.1 (fits_read_errmsgs), 3.2.2 (_fits_read_errmsg)`

3.2.2 _fits_read_errmsg

Synopsis

Retrieve an error message from the cfitsio error message stack

Usage

`String_Type _fits_read_errmsg ()`

Description

This function is a wrapper around the cfitsio library function `fits_read_errmsg`. See its documentation for additional information.

Notes

This function returns `NULL` if there are no error messages available.

See Also

`2.7.1 (fits_read_errmsgs), 2.7.2 (fits_set_verbose_errors)`

3.2.3 _fits_open_file

Synopsis

Open a fits file

Usage

`status = _fits_open_file (Ref_Type fptr, String_Type file, String_Type mode)`

Description

This function is a wrapper around the cfitsio library function `fits_open_file`. See its documentation for additional information.

The main difference between this function and its cfitsio counterpart is that the `mode` argument is a string whose value must be one of the following:

```
"r"    Open the file in read-only mode
"w"    Open the file in read-write mode
"c"    Create a new file.
```

Note that if the mode argument is "c", then the cfitsio `fits_create_file` function will be called. An important difference between this intrinsic function and the `fits_create_file` library function is that if the file already exists, the library function will return an error, whereas `_fits_open_file` will simply delete the file before creating a new one.

3.2.4 _fits_delete_file

Synopsis

Delete the file associated with a Fits_File_Type object

Usage

```
status = _fits_delete_file (Fits_File_Type fptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_delete_file`. See its documentation for additional information.

3.2.5 _fits_close_file

Synopsis

Close a fits file

Usage

```
status = _fits_close_file (Fits_File_Type fptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_close_file`. See its documentation for additional information.

3.2.6 _fits_movabs_hdu

Synopsis

Move to an absolute HDU number

Usage

```
status = _fits_movabs_hdu (Fits_File_Type fptr, Int_Type hdunum)
```

Description

This function is a wrapper around the cfitsio library function `fits_movabs_hdu`. See its documentation for additional information.

Notes

The cfitsio counterpart also returns the HDU type of the resulting HDU.

3.2.7 _fits_movrel_hdu

Synopsis

Move a relative number of HDUs

Usage

```
status = _fits_movrel_hdu (Fits_File_Type fptr, Int_Type nmove)
```

Description

This function is a wrapper around the cfitsio library function `fits_movrel_hdu`. See its documentation for additional information.

Notes

The cfitsio counterpart also returns the HDU type of the resulting HDU.

3.2.8 _fits_movnam_hdu

Synopsis

Move to a named HDU

Usage

```
status = _fits_movnam_hdu (fptr, hdutype, extname, extvers)
```

```
    Fits_File_Type fptr;
    Int_Type hdutype, extvers;
    String_Type extname;
```

Description

This function is a wrapper around the cfitsio library function `fits_movnam_hdu`. See its documentation for additional information.

3.2.9 _fits_get_num_hdus

Synopsis

Return the number of HDUs in a FITS file

Usage

```
status = _fits_get_num_hdus (Fits_File_Type fptr, Ref_Type hdunum)
```

Description

This function is a wrapper around the cfitsio library function `fits_get_num_hdus`. See its documentation for additional information.

3.2.10 _fits_get_hdu_num

Synopsis

Return the current HDU number

Usage

```
hdunum = _fits_get_hdu_num (Fits_File_Type fptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_get_hdu_num`. See its documentation for additional information.

3.2.11 _fits_get_hdu_type

Synopsis

Get the current HDU type

Usage

```
status = _fits_get_hdu_type (Fits_File_Type fptr, Ref_Type hdutype)
```

Description

This function is a wrapper around the cfitsio library function `fits_get_hdu_type`. See its documentation for additional information.

Upon a sucessful return, the value of the variable associated with the `hdutype` reference will be set to one of the following constants:

```
_FITS_IMAGE_HDU
_FITS_ASCII_TBL
_FITS_BINARY_TBL
```

3.2.12 _fits_copy_file

Synopsis

Copy a fits file

Usage

```
status = _fits_copy_file (infptr, outfptr, previous, current, following)

       Fits_File_Type infptr, outfptr;
       Int_Type previous, current, following;
```

Description

This function is a wrapper around the cfitsio library function `fits_copy_file`. See its documentation for additional information.

3.2.13 _fits_copy_hdu

Synopsis

Copy an HDU

Usage

```
status = _fits_copy_hdu (infptr, outfptr, morekeys)

       Fits_File_Type infptr, outfptr;
       Int_Type morekeys;
```

Description

This function is a wrapper around the cfitsio library function `fits_copy_hdu`. See its documentation for additional information.

3.2.14 _fits_copy_header

Synopsis

Copy a fits header from one HDU to another

Usage

```
status = _fits_copy_header (Fits_File_Type infptr, Fits_File_Type outfptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_copy_header`. See its documentation for additional information.

3.2.15 _fits_delete_hdu

Synopsis

Delete the current HDU

Usage

```
status = _fits_delete_hdu (Fits_File_Type fptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_delete_hdu`. See its documentation for additional information.

Notes

The corresponding cfitsio function also returns the HDU type of the new HDU. If that information is necessary, make a separate call to `_fits_get_hdu_type`.

3.2.16 _fits_create_img

Synopsis

Create a new image extension

Usage

```
status = _fits_create_img (fptr, bitpix, dims)

       Fits_File_Type fptr;
       Int_Type bitpix;
       Array_Type dims;
```

Description

This function is a wrapper around the cfitsio library function `fits_create_img`. See its documentation for additional information.

Notes

This function differs from the corresponding cfitsio function in that the `dims` array is assumed to be a 1-d integer array whose elements specify the number of axes and the size of each axis. In particular, the value of the NAXIS keyword will be given by `length(dims)`, the value of NAXIS1 will be given by `dims[0]`, and so forth.

3.2.17 _fits_write_img

Synopsis

Write an image

Usage

```
status = _fits_write_img (Fits_File_Type fptr, Array_Type img
```

Description

This function is a wrapper around the cfitsio library function `fits_write_img`. See its documentation for additional information.

Notes

This function differs from its cfitsio counterpart in that the whole image represented by the array `img` will be written out.

3.2.18 _fits_read_img

Synopsis

Read an image

Usage

```
status = _fits_read_img (Fits_File_Type fptr, Ref_Type img)
```

Description

This function is a wrapper around the cfitsio library function `fits_read_img`. See its documentation for additional information.

Notes

This function differs from the corresponding cfitsio routine in that the type of the image returned is automatically determined by the routine via a call to `fits_get_img_type`. The dimensionality of the returned image is given by `[...,NAXIS2,NAXIS1]` such that `NAXIS1` corresponds to the fastest varying dimension, `NAXIS2` the next fastest varying, etc.

3.2.19 _fits_create_binary_tbl

Synopsis

Create a binary table extension

Usage

```
status = _fits_create_binary_tbl (fptr, naxis2, ttype, tform, tunit, extname)
```

```
Fits_File_Type fptr;
Array_Type tunit, tform, ttype;
Int_Type naxis2;
String_Type extname;
```

Notes

The `_fits_create_binary_tbl` function is a wrapper around the `fits_create_tbl` function explicitly creating a binary table. The input arrays `ttype`, `tform`, and `tunit` must be the same length, which determines the number of columns in the table. The `tunit` and `extname` parameters may be set to `NULL`.

3.2.20 _fits_update_key

Synopsis

Update a keyword or append a new one

Usage

```
status = _fits_update_key (fptr, keyname, value, comment)
```

```
Fits_File_Type fptr;
String_Type keyname, comment;
Any_Type value;
```

Description

This function is a wrapper around the cfitsio library function `fits_update_key`. See its documentation for additional information.

Notes

The data-type for the `value` argument must be an approriate type for FITS keywords. If `value` is a string, then the string will be written as a cfitsio long-string using the fits_update_key_longstr function. If value is NULL, then the fits_update_key_null function will be called to update the keyword.

The comment parameter may be set to NULL to if the comment field associated with the keyword is not to be modified.

To update the value of a boolean (logical) keyword, use the _fits_update_logical function.

3.2.21 _fits_update_logical

Synopsis

Update the value of a boolean keyword

Usage

```
status = _fits_update_logical (fptr, keyname, value, comment)
```

```
Fits_File_Type fptr;
String_Type keyname, comment;
Int_Type value;
```

Description

The _fits_update_logical function is a wrapper around the cfitsio fits_update_key function with TLOGICAL specified as the datatype argument. If the value parameter is non-zero, then a value T (TRUE) will be given to the specified keyword. Otherwise, the value of the keyword will be set to F (FALSE).

If the comment parameter is NULL, then the keyword's comment field will not be modified.

See Also

3.2.20 (_fits_update_key)

3.2.22 _fits_write_comment

Synopsis

Write a COMMENT header

Usage

```
status = _fits_write_comment (Fits_File_Type fptr, String_Type comment)
```

Description

This function is a wrapper around the cfitsio library function fits_write_comment. See its documentation for additional information.

3.2.23 _fits_write_history

Synopsis

Write a HISTORY header

Usage

```
status = _fits_write_history (Fits_File_Type fptr, String_Type history)
```

Description

This function is a wrapper around the cfitsio library function `fits_write_history`. See its documentation for additional information.

3.2.24 _fits_write_date

Synopsis

Write a DATE keyword

Usage

```
status = _fits_write_date (Fits_File_Type fptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_write_date`. See its documentation for additional information.

3.2.25 _fits_write_record

Synopsis

Write a keyword record

Usage

```
status = _fits_write_record (Fits_File_Type fptr, String_Type card)
```

Description

This function is a wrapper around the cfitsio library function `fits_write_record`. See its documentation for additional information.

3.2.26 _fits_modify_name

Synopsis

Rename a keyword

Usage

```
status = _fits_modify_name (fptr, oldname, newname)

       Fits_File_Type fptr;
       String_Type oldname, newname;
```

Description

This function is a wrapper around the cfitsio library function `fits_modify_name`. See its documentation for additional information.

3.2.27 _fits_get_num_keys

Synopsis

Get the number of keywords in the current HDU

Usage

```
status _fits_get_num_keys (Fits_File_Type fptr, Ref_Type numkeys)
```

Description

This function is a wrapper around the cfitsio `fits_get_hdrspace` function. It obtains the number of existing keywords in the current HDU (excluding the END keyword) and assigns that value to variable associated with the `numkeys` parameter.

3.2.28 _fits_read_key_integer

Synopsis

Read the value of a keyword as an integer

Usage

```
status = _fits_read_key_integer (fptr, keyname, value, comment)

      Fits_File_Type fptr;
      String_Type keyname;
      Ref_Type value, comment;
```

Description

This function uses the cfitsio `fits_read_key` function to read the value of the specifed keyword as an integer. Its value is assigned to the variable referenced by the `value` parameter. If the comment parameter is non-NULL, then the value of the comment field will be assigned to it.

See Also

3.2.31 (_fits_read_key), 3.2.29 (_fits_read_key_string), 3.2.30 (_fits_read_key_double)

3.2.29 _fits_read_key_string

Synopsis

Read the value of a keyword as a string

Usage

```
status = _fits_read_key_string (fptr, keyname, value, comment)

      Fits_File_Type fptr;
      String_Type keyname;
      Ref_Type value, comment;
```

Description

This function uses the cfitsio `fits_read_key_longstr` function to read the value of the specifed keyword as a cfitsio long-string. The string is assigned to the variable referenced by the `value` parameter. If the comment parameter is non-NULL, then the value of the comment field will be assigned to it.

See Also

`3.2.31 (_fits_read_key), 3.2.28 (_fits_read_key_integer), 3.2.30 (_fits_read_key_double)`

3.2.30 _fits_read_key_double

Synopsis

Read the value of a keyword as a double

Usage

```
status = _fits_read_key_double (fptr, keyname, value, comment)

       Fits_File_Type fptr;
       String_Type keyname;
       Ref_Type value, comment;
```

Description

This function uses the cfitsio `fits_read_key` function to read the value of the specifed keyword as a double. The keyword's value is assigned to the variable referenced by the `value` parameter. If the comment parameter is non-NULL, then the value of the comment field will be assigned to it.

See Also

`3.2.31 (_fits_read_key), 3.2.28 (_fits_read_key_integer), 3.2.29 (_fits_read_key_string)`

3.2.31 _fits_read_key

Synopsis

Read the value of a keyword

Usage

```
status = _fits_read_key (fptr, keyname, value, comment)

       Fits_File_Type fptr;
       String_Type keyname;
       Ref_Type value, comment;
```

Description

This function uses the cfitsio `fits_read_key` function to read the value of the specifed keyword. It first uses the cfitsio `fits_get_keytype` function to determine the data-type for the keyword and then calls `fits_read_key` using that data-type. The resulting value is assigned to the variable referenced by the `value` parameter. If the comment parameter is non-NULL, then the value of the comment field will be assigned to it.

See Also

`3.2.28 (_fits_read_key_integer), 3.2.29 (_fits_read_key_string), 3.2.30 (_fits_read_key_double)`

3.2.32 _fits_read_record

Synopsis

Read a specified record from the current HDU

Usage

```
status = _fits_read_record (fptr, keynum, card)

       Fits_File_Type fptr;
       Int_Type keynum;
       Ref_Type card;
```

Description

This function is a wrapper around the cfitsio library function `fits_read_record`. See its documentation for additional information.

3.2.33 _fits_delete_key

Synopsis

Delete a keyword from the header

Usage

```
status = _fits_delete_key (Fits_File_Type fptr, String_Type keyname)
```

Description

This function is a wrapper around the cfitsio library function `fits_delete_key`. See its documentation for additional information.

3.2.34 _fits_get_colnum

Synopsis

Get the column number of a specfied table column

Usage

```
status = _fits_get_colnum (fptr, colname, colnum)

       Fits_File_Type fptr;
       String_Type colname;
       Ref_Type colnum;
```

Description

This function is a wrapper around the cfitsio library function `fits_get_colnum`. See its documentation for additional information.

Notes

The corresponding cfitsio function permits a wildcard match to the `colname` parameter. The current wrapping of this function does not support such matching.

The `colname` parameter is treating in a case-insensitive manner.

3.2.35 _fits_insert_rows

Synopsis

Insert rows into a table

Usage

```
status = _fits_insert_rows (fptr, firstrow, nrows)
```

```
Fits_File_Type fptr;
Int_Type firstrow, nrows;
```

Description

This function is a wrapper around the cfitsio library function `fits_insert_rows`. See its documentation for additional information.

3.2.36 _fits_delete_rows

Synopsis

Delete rows from a table

Usage

```
status = _fits_delete_rows (fptr, firstrow, nrows)
```

```
Fits_File_Type fptr;
Int_Type firstrow, nrows;
```

Description

This function is a wrapper around the cfitsio library function `fits_delete_rows`. See its documentation for additional information.

3.2.37 _fits_insert_cols

Synopsis

Insert columns into a table

Usage

```
status = _fits_insert_cols (fptr, colnum, ttype, tform)
```

```
Fits_File_Type fptr;
Int_Type colnum;
Array_Type ttype, tform;
```

Description

This function is a wrapper around the cfitsio library function `fits_insert_cols`. See its documentation for additional information.

Notes

The number of columns to be inserted is given by the length of the `ttype` and `tform` arrays, which must be of the same length.

3.2.38 _fits_delete_col

Synopsis

Delete a column from a table

Usage

```
status = _fits_delete_col (Fits_File_Type fptr, Int_Type colnum)
```

Description

This function is a wrapper around the cfitsio library function `fits_delete_col`. See its documentation for additional information.

3.2.39 _fits_get_num_cols

Synopsis

Get the number of table columns

Usage

```
status = _fits_get_num_cols (Fits_File_Type fptr, Ref_Type ncols)
```

Description

This function is a wrapper around the cfitsio library function `fits_get_num_cols`. See its documentation for additional information.

3.2.40 _fits_get_rowsize

Synopsis

Get the number of rows to read or write for maximum efficiency

Usage

```
status = _fits_get_rowsize (Fits_File_Type fptr, Ref_Type nrows)
```

Description

This function is a wrapper around the cfitsio library function `fits_get_rowsize`. See its documentation for additional information.

3.2.41 _fits_get_num_rows

Synopsis

Get the number of table rows

Usage

```
status = _fits_get_num_cols (Fits_File_Type fptr, Ref_Type nrows)
```

Description

This function is a wrapper around the cfitsio library function `fits_get_num_rows`. See its documentation for additional information.

3.2.42 _fits_write_col

Synopsis

Write data to a table column

Usage

```
status = _fits_write_col (fptr, colnum, firstrow, firstelem, array)

       Fits_File_Type fptr;
       Int_Type colnum;
       Int_Type firstrow, firstelem;
       Array_Type array;
```

Description

This function is a wrapper around the cfitsio library function `fits_write_col`. See its documentation for additional information.

Notes

The number of elements written out to the column by this function will be equal to the number of elements in the array.

3.2.43 _fits_read_col

Synopsis

Read elements from a column

Usage

```
status = _fits_read_col (fptr, colnum, firstrow, numrows, array

       Fits_File_Type fptr;
       Int_Type colnum, firstrow, numrows;
       Ref_Type array;
```

Description

This function is a complicated wrapper around a number of cfitsio functions to enable it to read any type of column, including vector and variable length columns. The data read by the function is assigned as the appropriately typed array to the variable referenced by the `array` argument.

For ordinary scalar columns, a 1-d array of size `numrows` will be produced. For a vector column, a 2d array of size `[numrows,repeat]` will be generated. Here `repeat` is given by the repeat value associated with the column. For a variable length column, where data are stored in the heap of the HDU, the data will be read as a 1-d array of `numrows` arrays.

If the column is a bit-valued column, then data will be returned as an array of integers of the appropriate size. Currently only 8X, 16X, and 32X bit columns are supported.

See Also

`3.2.46 (_fits_read_cols), 3.2.42 (_fits_write_col)`

3.2.44 _fits_get_keytype

Synopsis

Get a keyword's data type

Usage

```
status = _fits_get_keytype (fptr, keyname, type)
```

```
Fits_File_Type fptr;
String_Type keyname;
Ref_Type type;
```

Description

This function is a wrapper around the cfitsio library function `fits_get_keytype`. See its documentation for additional information.

Notes

This function differs from its cfitsio counterpart in that instead of explicitly specifying the keyword's value string, this function uses the value of the specified keyword. It also returns the type as a **S-lang** `DataType_Type` object, e.g., `Int_Type`, `Complex_Type`, etc.

3.2.45 _fits_get_keyclass

Synopsis

Get the class of an input header record

Usage

```
Int_Type _fits_get_keyclass (String_Type card)
```

Description

This function is a wrapper around the cfitsio library function `fits_get_keyclass`. See its documentation for additional information.

3.2.46 _fits_read_cols

Synopsis

Read one or more table columns

Usage

```
status = _fits_read_cols (fptr, colnums, firstrow, nrows, arrays)
```

```
Fits_File_Type fptr;
Array_Type colnums;
Int_Type firstrow, numrows;
Ref_Type arrays;
```

Description

This function performs a similar task as the `_fits_read_col`. The main difference is that instead of reading a single column, it is capable of reading multiple columns specified by the `colnums` parameter. It assigns the data as an array of arrays to the variable referenced by the `arrays` parameter. See the documentation for the `_fits_read_col` function for more information.

Notes

This function takes advantage of the cfitsio buffering mechanism to optimize the reads.

3.2.47 _fits_write_chksum

Synopsis

Compute and write DATASUM and CHECKSUM keywords

Usage

```
status = _fits_write_chksum (Fits_File_Type fptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_write_chksum`. See its documentation for additional information.

3.2.48 _fits_update_chksum

Synopsis

Update the CHECKSUM keyword

Usage

```
status = _fits_update_chksum (Fits_File_Type fptr)
```

Description

This function is a wrapper around the cfitsio library function `fits_update_chksum`. See its documentation for additional information.

3.2.49 _fits_verify_chksum

Synopsis

Verify the checksums for the current HDU

Usage

```
status = _fits_verify_chksum (fptr, dataok, hduok)
```

```
Fits_File_Type fptr;
Ref_Type dataok, hduok;
```

Description

This function is a wrapper around the cfitsio library function `fits_verify_chksum`. See its documentation for additional information.

3.2.50 _fits_get_chksum

Synopsis

Get the checksums for the current HDU

Usage

```
status = _fits_get_chksum (fptr, datasum, hdusum)
```

```
Fits_File_Type fptr;
Ref_Type datasum, hdusum;
```

Description

This function is a wrapper around the cfitsio library function `fits_get_chksum`. See its documentation for additional information.

3.2.51 _fits_get_version

Synopsis

Get the cfitsio library version number

Usage

```
Float_Type _fits_get_version ()
```

Description

This function is a wrapper around the cfitsio library function `fits_get_version`. See its documentation for additional information.