*Simple and Light Interfaces for C and C++ users*

# SFITSIO ユーザーズリファレンスガイド

**Version 1.2.1　日本語版**

## Credits


Software Development:

*Chisato Yamauchi*

Manual Document:

*Chisato Yamauchi*, *Ken Ebisawa* and *IMC LTD.*

Special Thanks:

*Daisuke Ishihara*, *Hajime Baba*, *Keiichi Matsuzaki* and *Yukio Yamamoto*


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

SFITSIO は JAXA の資産であり，ISAS/JAXA データ利用 G の公式サポートソフトウェアです

# 目 次

# 1 はじめに

## 1.1 ご愛用の FITS I/O ライブラリを使って，この課題を何分で解答できますか?

**課題 1**

複数のバイナリテーブル HDU を持つ FITS ファイル「foo.fits.gz」があります．それを読み込んで，バイナリテーブル「A」のカラム「X」の最初の行の値が NULL 値ではなく，かつバイナリテーブル「B」のカラム「Y」の最初の行の値が 0 を越える場合に，変数 flag に 1 をセットするコードを書いてください．ただし，ヘッダには TZERO$n$，TNULL$n$ がセットされているのでその事も考慮する必要があります．なお，各種エラーチェックと#include から main 関数までのコードは省略してもかまいません．

**SFITSIO を使った課題 1 の解答例**

```
fitscc fits;
long row_idx = 0;
double v;
fits.read_stream("foo.fits.gz");                                  /* ファイルを読む */
if ( isfinite(fits.table("A").col("X").dvalue(row_idx)) &&     /* 値をテスト */
     isfinite(v=fits.table("B").col("Y").dvalue(row_idx)) && 0 < v ) flag = 1;
```

**課題 2**

4 個の CCD チップによる観測データを保存した「obj0.fits」があります．1 つの CCD の解像度は 1024×2048 で，4 個分が obj0.fits の Primary HDU に 4096×2048 の 1 枚の画像として保存されています．この FITS ファイルを読み込み，CCD ごとに HDU を分割した別の FITS ファイルを作成してください．HDU の名前はそれぞれ,「CCD0」「CCD1」…のようにします．ただし，ヘッダには BZERO がセットされているのでその事も考慮する必要があります．なお，各種エラーチェックと#include から main 関数までのコードは省略してもかまいません．

**SFITSIO を使った課題 2 の解答例**

```
const char *name[] = {"CCD0","CCD1","CCD2","CCD3"};
fitscc fits0, fits1;                                  /* 元 FITS と 新しい FITS */
fits0.read_stream("obj0.fits");                       /* 観測フレームを読む */
fits_image &pri0_img = fits0.image("Primary");        /* 長いので alias を定義 */
for ( i=0 ; i < 4 ; i++ ) {                           /* HDU を追加し，コピー */
  fits1.append_image(name[i],0, pri0_img.type(), 1024,2048);
  pri0_img.copy(&(fits1.image(i)), 1024*i, 1024, 0, 2048);
}
fits1.write_stream("obj0_separated.fits");            /* 別のファイルに保存 */
```

**課題 3**

処理済みのバイアスフレーム「bias.fits.gz」，フラットフレーム「flat.fits.gz」と，未処理の観測フレーム「obj1.fits」があります．観測フレームについて，オーバスキャン (平均値) 引き，バイアス引き，フラット補正をした後，画像の上 1/4 と下 1/4 を削除した FITS を作成してください．オーバスキャンの領域情報は，ヘッダの BIASSEC キーワードの値に IRAF 準拠の形式で保存されています．ただし，観測フレームは 16-bit 整数型で保存され，ヘッダには BZERO がセットされているのでその事も考慮する必要があります．なお，各種エラーチェックと#include から main 関数までのコードは省略してもかまいません．

**SFITSIO を使った課題 3 の解答例**

```
fitscc bias_fits, flat_fits, obj1_fits;
long y_len, sect[4];
double v_stat[1];
bias_fits.read_stream("bias.fits.gz");                  /* バイアスを読む */
flat_fits.read_stream("flat.fits.gz");                  /* フラットを読む */
obj1_fits.read_stream("obj1.fits");                     /* 観測フレームを読む */
fits_image &obj1pri = obj1_fits.image("Primary");       /* 長いので alias を定義 */
obj1pri.convert_type(FITS::DOUBLE_T);                   /* データ型を変換する */
obj1pri.header("BIASSEC").get_section_info(sect,4); /* overscan 情報を取得 */
obj1pri.stat_pixels(v_stat,1, "results=mean",           /* overscan 領域の */
                    sect[0],sect[1],sect[2],sect[3]); /* 平均値を求める */
obj1pri.subtract(v_stat[0]);                            /* overscan(平均値) 引き */
obj1pri.subtract(bias_fits.image("Primary"));           /* バイアス引き */
obj1pri.divide(flat_fits.image("Primary"));             /* フラット補正 */
y_len = obj1pri.row_length();                           /* 画像の縦の長さ */
obj1pri.data_array().crop(1,y_len/4,y_len/2);           /* 上下 1/4 を削除 */
obj1_fits.write_stream("obj1_done.fits");               /* 保存 */
```

SFITSIO を使ったサンプルプログラムは §13 APPENDIX1 でも紹介しています．参考になると思いますのでぜひそちらもご覧ください．

## 1.2 なぜ SFITSIO が誕生したのか?

SFITSIO は，ISAS/JAXA の赤外線天文衛星「あかり」のデータ解析ソフトウェアプロジェクトにおいて開発が始まりました．

このプロジェクトでは，全てのデータプロセッシング用の標準データフォーマットとして，観測機

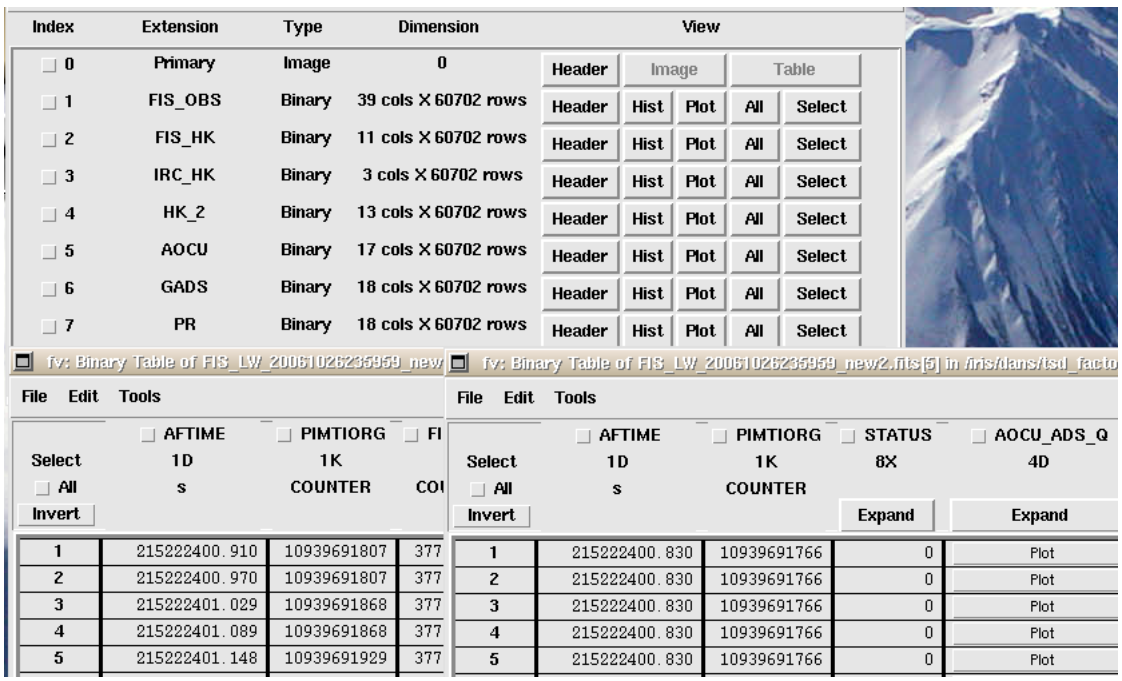



図 1: 赤外線天文衛星「あかり」のソフトウェアプロジェクトで使われている標準データフォーマット「TSD」を fv で開いた様子．Primary 以外はすべてバイナリテーブルである．

器，衛星本体，軌道情報等の多岐にわたる時系列データを複数のバイナリテーブル HDU に詰めて 1 つの FITS とする「Time Series Data(以降 TSD)」と呼ばれる FITS ファイルを定義しました (図 1). これはすなわち,「あかり」の全天サーベイ用ツール群，各種ポインティング観測用ツール群といったあらゆるデータ解析ツールは，この TSD が必ず最初の入力ファイルになるという事を意味します. しかし，図 1のとおり，この TSD は多数のデータ型・様々なデータ格納手法からなるデータ構造を持つ複雑な FITS ファイルです．したがって，プロジェクトでは，各々の解析チームが勝手な方法で TSD にアクセスするコードを開発するのは合理的ではないと判断し，TSD に対して効率的にアクセスできる標準ライブラリを開発する事にしました.

プロジェクトで開発した TSD 用の標準ライブラリ (C 言語，IDL)[1] では，FITS ファイルの内容はオンメモリ[2] で扱い，**FITS の構造をそのまま表現した API** を基本としました[3]．その結果，C 言語の場合は次のようなコードで TSD にアクセスできるようになりました.

```
tsd = fits_cache__open(class_level, NULL, "tsd_xxx.fits");  /* ファイルを読む */
(省略: hdu_index, col_index をセットするためのコードなど)
ptr = tsd->btables[hdu_index]->columns[col_index].data;   /* カラムにアクセス */
```

これは，最初の行で FITS ファイルをメモリ上にロードし，`ptr = ...` の行のコードではテーブルのカラムデータの先頭アドレスを得るものですが,「`tsd->`」から順に，FITS 全体，バイナリテーブル HDU，カラム，のように API が FITS の構造そのものになっています (構造体で表現)．このレベルの API なら，様々なバイナリテーブル HDU にアクセスする場合でもそれほど苦労せずにコードを書けますし，そこそこの可読性も確保できます．しかし，このライブラリは，上記のように必ずメモリに直接アクセスするため安全とは言えず，`TZERO`$n$，`TSCAL`$n$ の計算はプロジェクトでは不要だったために省略されているなど，汎用 FITS ライブラリとしては不十分な点が多いものでした.

上記の C ライブラリにおける考え方を原点とし，その開発経験を踏まえて一から作りなおしたものが，SFITSIO です．SFITSIO では**構造体の上位互換である** C++の**「クラス」**を使い，Binary Table HDU については上記 C ライブラリで見られた諸問題を一挙に解決する事に成功し，Image HDU についても直感的な API による画像解析をサポートする事で，より洗練された汎用 FITS ライブラリへ進化しました．§1.1の課題 1 の解答をご覧ください．ポインタ変数を使わずに，テーブルやカラムの名前を使ってバイナリテーブルのセルの値にアクセスするためのコードが，瞬間的に書けるようになっています．課題 2 と課題 3 は Image HDU に関する処理を行なうものですが,「やりたい事が，わかりやすいコードで，簡単に書ける」という事がおわかりいただけると思います.

「あかり」プロジェクトの「**データプロセッシングにおいて，FITS の I/O を可能な限り脇役に追いやる**」という方針は，どのプロジェクトでも同じはずです．近年では装置の高性能化や大規模サーベイによって，ますますデータプロセッシングの複雑さが増しており，汎用的で高性能かつ高レベルな (目的がすぐに達成できる)FITS I/O ライブラリが切望されています.

SFITSIO はこの難しい課題に対する有用な解を提供すべく開発されてきました．今後のバージョンアップにおいては，現代の計算機環境の進化に応じた新しい解を提案していく予定です.

---

[1] IDL 向けには馬場肇氏によって TSD 専用のインタフェースが作られました．非常に理にかなった設計と美しいコードで，SFITSIO はこれにかなり影響されています.

[2] 広大なメモリ領域が使える 64-bit Linux が爆発的に普及した事もあり,「オンメモリ」という方針も大正解でした.

[3] 厳密には,「オンメモリ」は「FITS の構造を表現した API」の実現には必須ではありません．しかし，ディスクベースでの実装は開発コストが高くつきすぎるので，オンメモリが現実的な解と言えるでしょう.

## 1.3 SFITSIO がプログラマの負担を最小化できるこれだけの理由

### 1.3.1 C++の作法を強要しない—C 言語の知識で使える!

SFITSIO は C++で書かれているので，SFITSIO の API を使ったコードは C++コンパイラでコンパイルします．と言われて「なんだ，C++を勉強しないといけないのか」と思ったあなた．その必要はありません．C++は C の上位互換ですから，C++コンパイラでも，C 言語のコード

```
#include <stdio.h>
int main()
{
    printf("Hello World\n");
}
```

がそのままコンパイルできます．一般的な C++入門書には「`cout << "foo" << endl;`」といったコードが出てきますが，**SFITSIO の利用にはそのような C++的作法の習得は不要です**[4]．これまで通り，C 言語の知識を使って C 言語の書き方でコードを書く事ができます．もちろん，このマニュアルも C 言語の書き方で作成していますので，ご安心ください．

### 1.3.2 メモリ管理の自動化により，恐怖のメモリリークとサヨナラ!

C 言語では，`malloc()` で確保したメモリ領域は必ず `free()` で開放しなければなりません．もし，`free()` を書き忘れると，メモリリークが発生します．コード量が小さい場合はメモリリークが発生してもなんとか修正できるものですが，大規模プログラムにおいてはメモリリークを 100% 阻止する事や，実際に起こった場合の対応は困難を極めます．大規模プログラムのメモリリーク問題はそういう点で，原発における放射性物質の閉じ込めと似ています．

このような「漏れ出し」問題をできる限り起こさないようにするため，SFITSIO では C++の「クラス」を使って FITS に関するメモリ管理を自動化しています．したがって，プログラマによる `malloc()`，`free()` は基本的には不要になるため，FITS を扱う上でのメモリリークは起きません[5]．例えば，SFITSIO で FITS ファイルを読む場合，次のような流れになります．

(1) プログラマが SFITSIO の API を使い，SFITSIO に FITS ファイルを読むよう指示する (次の例は，`myimage.fits.gz` を読む場合)．

```
fits.read_stream("myimage.fits.gz");
```

(2) SFITSIO は FITS ファイルを解析し，必要なメモリ領域を確保する．

(3) SFITSIO は確保したメモリ上に FITS ファイルの内容すべて (または一部) をコピーする．

(4) プログラマが SFITSIO の API を使ってメモリ上の必要なデータにアクセスする (次の例はヘッダの `CRVAL1` を読む場合)．

```
printf("crval1 = %f\n", fits.image("Primary").header("CRVAL1").dvalue());
```

(5) スコープを抜けたら，確保されたメモリ領域は自動的に開放される．

プログラマは (1)，(4) のコードを書くだけで済む上に，メモリリークの心配もありません．

メモリの管理はライブラリに任せて，メモリリークと格闘するのはもう終わりにしましょう．

---

[4] google 社のコーティング規約では，`cout << ...` は使うべきでないとされ，`printf()` の使用を推奨しています．

[5] `new` を使う場合は可能性がありますが，使わなければならない場面は少ないものです．

### 1.3.3 API は FITS の構造そのもの—だから一度使ったらもう忘れない!

どうして SFITSIO の API は一度使ったら忘れないのか?—その秘訣は **API の形**にあります.
では，SFITSIO の API とはどんなものなのか? それを，次の 3 つの例でみていきましょう.

(1) プライマリ HDU のヘッダ CRVAL1 の値を読んで出力する例:

```
printf("crval1 = %f\n", fits.image("Primary").header("CRVAL1").dvalue());
```

(2) プライマリ HDU の画像の座標 (x,y) のピクセル値を読んで出力する例:

```
printf("pixel = %f\n", fits.image("Primary").dvalue(x,y));
```

(3) 「CATALOG」というバイナリテーブル HDU の，カラム「RA」の最初の行の値を読んで出力する例:

```
printf("ra = %f\n", fits.table("CATALOG").col("RA").dvalue(0));
```

いかがでしょうか? ピンときたでしょうか?
そうです．API は「FITS の構造そのまんま」なのです．例えば 3 つめの例では,

FITS の テーブル の カラム の double 値

となっています.「の」を「.」に置き換え，FITS の構造のとおりに関数名を並べれば自動的に API ができあがります．しかも SFITSIO の場合は，覚えなければならない関数名は極めて限られています.
まず，構造物として必須の関数名は次の 5 つしかありません:

| hdu() | image() | header() | table() | col() |
|---|---|---|---|---|
| 任意の HDU | Image HDU | FITS ヘッダ | ASCII/Binary Table HDU | テーブルのカラム |

これに加え，値を読み出すための関数名 dvalue()，lvalue()，llvalue()，svalue() (それぞれ，double 型，long 型，long long 型，文字列型を返す) と，値を書き込むための関数 assign() を覚えれば，あらゆる FITS の基本的な読み書きができるようになります.
なお，dvalue() 等の関数は，FITS ファイルが持つ値が文字列であろうと整数値であろうと適切に変換したものを返します．逆に assign() では，引数の型によって[6] 適切に変換して FITS に値を格納します．しかも，これらの関数はライブラリ側でキャッシュしているヘッダの BZERO，BSCALE，TZERO$n$，TSCAL$n$ の値による変換もやってくれますから，プログラマはヘッダを読み取るコードを書く必要はありません．もちろん，**符号なし整数値**についても，BZERO または TZERO$n$ を適切にセットし，lvalue()，assign() を使うだけで読み書きができます.

### 1.3.4 複数の HDU 間のランダムアクセスも楽々

SFITSIO は，オンメモリ型のライブラリであり，プログラマによるファイルのシークも不要，メモリの管理も不要です．§1.3.3で述べたように API の形は FITS の構造そのものですから，複数の HDU 間のランダムアクセスを行なう場合でも，§1.1の解答例のように**瞬間的に**コードが書けます.

[6] C++では，同じ関数名で異なる引数を持つ関数を作る事ができます.

### 1.3.5 型変換，ZERO，SCALE，NULL 処理に対応した “高レベル API” の提供

§1.3.3，§1.3.4で紹介した，dvalue()，assign() 等の関数を,「高レベル API」と呼び，SFITSIO ではこれらをプログラマが通常使用するものと位置付けています.

これらの高レベル API では，型の変換や ZERO，SCALE の変換，さらには BLANK 値・NULL 値の変換も行ないます．SFITSIO では，これらの API をできるだけ性能を落とさないように気をつかって実装していますが，ピクセルあたりの演算が少なく，かつ大量のピクセルを扱う場合には関数呼び出しのオーバヘッドが問題になる事があります．そのような場合には，この後に紹介する低レベルな API や画像処理用 API を使う事ができます.

### 1.3.6 高速処理のための “低レベル・超低レベル API” の提供

高レベル API では十分なパフォーマンスが得られない場合は，ZERO，SCALE，NULL 処理を行なわない「低レベル API」，さらには SFITSIO が管理している内部メモリバッファのアドレスを取得可能な「超低レベル API」を使って，高速化を行なう事ができます.

高レベル API，低レベル API とをうまく使いわける事ができれば，コストパフォーマンスに優れたソフトウェア開発が可能です.

### 1.3.7 画像処理用 API の提供

近年では，サイズの大きい FITS 画像の解析も珍しくありません．SFITSIO には高速に動作する画像処理用の API があます．それらを使うと，少ないコードで領域のスキャン，コピー&ペースト，スカラー値・画像に対する四則演算，ユーザ関数による画像スキャンや画像演算が可能です.

### 1.3.8 様々な API で printf() 互換の可変長引数をサポート

printf() 関数で使われるフォーマット+可変長引数の仕様は，LIBC の文字列操作のための機能の中でも最強の利便性を誇るものです．SFITSIO では，printf() 関数と完全互換な引数をとる事ができる，colf()，headerf() などの「f」サフィックス付きの関数をいたるところに配置し，コーディングの手間を減らす工夫をしています．例えば，headerf() を使うと，

```
fits.image("Primary").headerf("CRVAL%d",i).dvalue()
```

のようにコードを書く事ができます (i は int 型の変数です).

### 1.3.9 圧縮ファイルや Web サーバ，ftp サーバ上のファイルの透過的扱い

SFITSIO では，ローカルディスクのファイル，あるいは Web サーバ，FTP サーバ上のファイルに対しても，gzip 形式，bzip2 形式の圧縮・伸長を自動的に行ないます (特別な API を使う必要はありません).

gzip 形式は他の FITS I/O ライブラリでも当然のように対応していますが，SFITSIO ではさらに，コマンドのインストールを行なうだけで fpack 等のあらゆる圧縮フォーマットに対応が可能です.

### 1.3.10 FITS ヘッダのコメント辞書の搭載と FITS テンプレート機能

SFITSIO は，FITS 規約や WCS で定義されたキーワードと，一般的によく使われるキーワードに対するコメント辞書を搭載しています．したがって，もはやプログラマが毎回毎回 FITS ヘッダのコメント文すべてを用意する必要はありません．例えば，

```
fits.image("Primary").header_fill_blank_comments();
```

のようにすると，プライマリ HDU のコメントが書かれていないヘッダレコードを，辞書が持つコメント文で一発で埋める事ができます．

さらに SFITSIO は，CFITSIO と同様のテンプレートファイル (§8) もサポートしています．テンプレートファイルとは，FITS ヘッダに類似した曖昧な文法で FITS の内容を定義できるテキスト形式のファイルで，これをもとにデータが入っていない新規 FITS ファイル (オブジェクト) を作成する事ができます．これにより，汎用的なツールの開発が容易に行なえます．テンプレートから FITS を作成する場合においても，テンプレートにコメントが書かれていない部分は自動的にコメントが与えられるので，テンプレートを作成するユーザも最低限の内容を書くだけで済みます．

もちろん，SFITSIO に搭載しているコメント辞書の内容が不十分な場合は，プログラマが辞書の内容を追加・変更できます．

### 1.3.11 超便利な s++スクリプト

SFITSIO を使ったコードから実行ファイルを作るためには，いくつかの標準的なライブラリをリンクする必要がありますが，付属の s++スクリプトを使えば，

```
$ s++ my_program.cc -lsfitsio
```

とするだけでコンパイルは完了します．また，引数のソースファイルが存在しない場合は，`#include` 〜main 関数まで書かれたソースのテンプレートファイルを作成する事ができますから，毎回毎回同じコードを書く手間を省く事ができます．ちょっと使いたい時に s++でパパっと作れてしまうのも，SFITSIO のありがたいところです．

### 1.3.12 CFITSIO ライクなディスクベースでの FITS I/O も OK

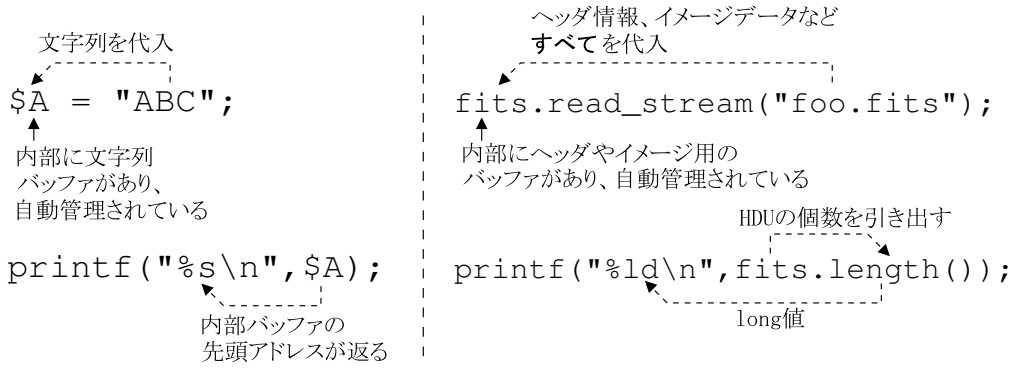
例えば，

- メモリをできるだけ使わずに，ヘッダだけ高速にスキャンしたい．
- 多くの画像ファイルのメディアンをとりたい．

といった場合には，FITS の内容をすべてメモリに取り込むのは得策ではありません．

SFITSIO では，1 つの Header Unit だけをメモリにロードしたり，1 つの Data Unit を読み飛ばすための API を提供しています．これらの API を使うには，まずプログラマがストリームハンドラを宣言して FITS ファイルをオープンします．次に，ストリームハンドラを目的のメンバ関数に与える事により，SFITSIO は FITS ファイルを必要な分だけスキャンします．例えば，ヘッダをロードするためのメンバ関数を使うと，ストリームの位置は続く Data Unit の先頭にセットされます．この場合でも，ヘッダ関連の高レベル API はそのまま使えますから，画像のアクセス程度ならディスクベースでも意外と簡単です．もちろん，シークが可能なストリームでは，ディスクアクセスに凝る事もできます．

**コラム: スクリプト言語と似ている SFITSIO のメモリバッファ管理**




スクリプト言語の変数に文字列を代入する事は，みなさんも日常的に行なう事だと思います．実は裏ではスクリプトエンジンがメモリの領域を計算し，確保し，管理しているわけですが，みなさんはそのような事をいちいち意識する事はないと思います．

一方，CFITSIO をはじめとする従来の手続き型 FITS I/O ライブラリの場合，ヘッダの文字列やデータ領域のためにプログラマがメモリ領域を計算し，`malloc()` 関数等を使って領域を確保し，不要になったら `free()` 関数で開放する，という煩わしい作業を繰り返し行なってきました．

§1.3.2で述べたように，SFITSIO では「クラス」を使って FITS ファイルの扱いをスクリプト言語の変数に近いレベルまで自動化しています．上の図は，perl と SFITSIO とを比較したもので，perl では `"ABC"` という文字列を変数に代入しており，SFITSIO では，`"foo.fits"` という FITS ファイル全体を fits という変数 (オブジェクト) に代入しています．この時，perl の場合も SFITSIO の場合も，必要なメモリ領域は自動的に確保，管理されます．FITS ファイルにはヘッダやイメージデータなど様々な情報が入っているため，変数 (オブジェクト) からそれらの情報を引き出したり (図の場合は，HDU の個数を引き出しています)，逆に情報を押し込めたりする時に，関数 (メンバ関数と呼びます) を使わなければならないのでスクリプト言語の変数に比べると手動操作的な部分がありますが，かなり perl と似た雰囲気で FITS ファイルが扱えるようになる事がおわかりいただけると思います．

このように比較すると，SFITSIO は特別な事をしているわけではない，とみる事ができるのではないでしょうか．

## 1.4 FAQ

- サポート体制はどうなっているのでしょう?

  SFITSIO と SLLIB は ISAS/JAXA データ利用 G の「公式サポートソフトウェア」であり，山内とデータ利用 G とが共同で開発・維持しています．ただし，形式上は MIT ライセンスで提供するオープンソースソフトウェアであり，無保証です．

- 使用実績はありますか?

筆者が把握している範囲では，JAXA 内での使用実績があります．具体的には,「あかり」の全天マップ作成プロジェクトと，赤外分野，月惑星科学分野での FITS 関連ツールの開発，ASTRO-H 向けの L1TSD プロジェクトなど，ほぼ全分野でのアーカイブ開発で使われています．

- C++のライブラリのようですが，上級者にしか扱えないのでは?

  そんな事はありません．むしろその逆で，従来，初級者には難しかった処理が SFITSIO なら簡単に行なえます．

  SFITSIO では，天文学の業界において平均的なスキルを持つ方を混乱させるような C++特有の技術の採用を避け，真に有用な C++技術のみを厳選して採用しています．さらにこのマニュアルは，C 言語の経験しか無い方を対象として作成しています．したがって，SFITSIO の利用には，C++の経験は不要です．

- 動作速度はどうですか?

  SFTISIO には，高レベル API，低レベル API，超低レベル API があり，どれを使うかに依存します．前者ほど少ない開発工数で済みますが動作は遅く，後者ほど開発工数が増えますが動作は速いです．さらに，画像処理用の API があり，それらの API が目的に合致する場合には，高速に動作するものが少ない工数で開発できます．

  原則として，ピクセル等のサイズを返すものと超低レベル API については，インラインメンバ関数です．これらについては，関数呼び出しのオーバヘッドは最小化されています．インラインメンバ関数は，SFITSIO のバージョンアップにあわせて，徐々に増やしていく予定です．

  さらに，SFITSIO は Intel® C++ Compiler でもビルドできます．

- 16-bit 符号無し整数は扱えますか?

  はい．高レベル API と画像処理用の API は `BZERO` 値の変換に対応しています．これらを使う場合は，プログラマ側での値の変換処理は不要です．16-bit 符号無し整数を新しい FITS に保存する場合，`BZERO` の値を 32768 にセットするだけです (§5.9を参照)．

  もし，高レベル API では速度が問題になる場合は，全ピクセル (要素) を任意の型に高速変換してくれる convert_type() メンバ関数 (§12.6.13，§12.8.28) と，超低レベル API とを併用すると簡単です．§5.11にコードの例を示していますのでご覧ください．

# 2 インストールと SFITSIO の始め方

## 2.1 対応 OS

SFITSIO が対応する OS は，Linux，FreeBSD，MacOSX，Solaris，Cygwin です．いずれも，32 ビット版と 64 ビット版の両方をサポートしています．

必要なコンパイラは GCC g++バージョン 3 系列以降です (筆者は 3.3.2 以降で動作確認を行なっています)．

## 2.2 SFITSIO のビルドと設置

SFITSIO のビルドの前に，SLLIB(Script-Like C-language library) version 1.2.1 以降をインストールする必要があります．SLLIB には，zlib，bzlib，readline ライブラリが必要ですので，あらかじめインストールしておきます (rpm の名前は，zlib-devel，bzip2-devel，readline-devel 等になっている事が多いようです[7])．

SLLIB をインストールしていない場合は，以下の手順でインストールします．SLLIB のアーカイブを展開して，make します[8]．

```
$ gzip -dc sllib-x.xx.tar.gz | tar xvf -
$ cd sllib-x.xx
$ make
```

ここで，64-bit 用のライブラリや 32-bit 用のライブラリを作りたい場合は，

```
$ make CCFLAGS="-m64"
```

のように，コンパイラに対してオプションを追加できます．32-bit OS 上の gcc の場合，SSE2 が有効にならない事があります[9]．その場合は次のように SSE に関係するオプションを与えるとパフォーマンスを改善できます．

```
$ make CCFLAGS="-msse2 -mfpmath=sse"
```

SLLIB をインストールします．

```
$ su
# make install32
```

この例は 32-bit OS の場合で，64-bit OS の場合は「`make install64`」とします．デフォルトでは「install32」では`/usr/local/lib`に,「install64」では`/usr/local/lib64` (Solaris の場合は`/usr/local/lib/64`) に `libsllib.a` がインストールされます．同時に，`/usr/local/include/sli` にヘッダファイル一式がコピーされ，`/usr/local/bin` に g++コンパイラの wrapper スクリプト `s++`がインストールされます．

次に，SFITSIO を同様の手順でインストールします．SFITSIO のアーカイブを展開して，make します[10]．

```
$ gzip -dc sfitsio-x.xx.tar.gz | tar xvf -
$ cd sfitsio-x.xx
$ make
```

---

[7] Debian 系の場合，zlib1g-dev，libbz2-dev，libreadline5-dev のようなパッケージ名になっています．

[8] 「`make shared`」で共有ライブラリを作る事もできます．ただし，よくわかってる方のみご利用ください．

[9] 64-bit OS ではデフォルトで有効になります

[10] 「`make shared`」で共有ライブラリを作る事もできます．ただし，よくわかってる方のみご利用ください．

SLLIB の場合と同様，次のようにコンパイラに対してオプションを追加できます.

```
$ make CCFLAGS="-m64"
```

```
$ make CCFLAGS="-msse2 -mfpmath=sse"
```

もし，make でエラーが報告された場合，`Makefile` の `INCDIR` に，SLLIB のヘッダファイルをインストールしたディレクトリを指定してください.

SFITSIO をインストールします.

```
$ su
# make install32
```

この例は 32-bit OS の場合で，64-bit OS の場合は「`make install64`」とします．デフォルトでは「install32」では`/usr/local/lib` に,「install64」では`/usr/local/lib64` (Solaris の場合は`/usr/local/lib/64`) に `libsfitsio.a` がインストールされます．同時に，`/usr/local/include/sli` にヘッダファイル一式がコピーされます.

以上でインストール作業は完了です.

## 2.3 マルチスレッド対応圧縮・伸長ツールのインストール (オプション)

SFITSIO-1.2.0 以降では，pigz (`http://zlib.net/pigz/`) または lbzip2 (`http://lacos.hu/`) がインストールされている環境では，ローカルの gzip 形式または bzip2 形式のファイルについて，自動的にマルチスレッドによる圧縮・伸長が行われます．これによって，マルチコアを持つ CPU が使える場合は圧縮・伸長にかかる時間を短くする事が可能です (特に**圧縮にかかる時間は劇的に小さくなります**)．必要に応じて pigz または lbzip2 をインストールします[11)].

## 2.4 サンプルプログラムを使った動作確認

次のコードは，CFITSIO のユーザーズガイドの第 2 章に含まれる「Example Program」を SFITSIO で書き直してみたものです[12)].

[11)] SFITSIO のインストールの前でも後でもかまいません

[12)] CFITSIO のサンプルコードと比較してみると，コードがすっきりしているのがおわかりいただけると思います.

```
#include <stdio.h>
#include <sli/fitscc.h>        /* required by every source that uses SFITSIO */
using namespace sli;

int main( int argc, char *argv[] )
{
    fitscc fits;                  /* fitscc object that expresses a FITS file */
    const long width = 300, height = 300;                    /* size of image */
    long len_written, i, j;

    /* Create the Primary HDU */
    fits.append_image("Primary",0, FITS::SHORT_T, width,height);
    fits_image &pri = fits.image("Primary");

    /* Add an "EXPOSURE" header record */
    pri.header("EXPOSURE").assign(1500.).assign_comment("Total Exposure Time");

    /* Set the values in the image with a linear ramp function */
    for ( j=0 ; j < height ; j++ ) {
        for ( i=0 ; i < width ; i++ ) pri.assign(i + j, i,j);
    }

    /* Output a FITS file */
    len_written = fits.write_stream("testfile.fits");
    printf("saved %ld bytes.\n", len_written);

    return (len_written < 0 ? -1 : 0);
}
```

では，s++コマンドを使ってコンパイルしてみましょう.

```
$ s++ fitsout.cc -lsfitsio
g++ -I/usr/local/include -L/usr/local/lib -Wall -O fitsout.cc -o fitsout -lsfits
io -lsllib -lz -lbz2 -lreadline -lcurses
$ ./fitsout
```

## 2.5 SFITSIO の始め方

- **お急ぎの方・とにかく試したい場合**

  §4.7「目的や API レベルごとのメンバ関数一覧」にざっと目を通してから，§5「チュートリアル」に進んでください.

- **FITS に精通されている方**

  §4「SFITSIO のクラス構造による FITS データ構造の表現と API」に進んでください.

- **FITS をよく知らないのでこの機会に勉強したいという方**

  §3「FITS のデータ構造の復習」を良く読んだ後，§4，§5に進んでください.

# 3 FITS のデータ構造の復習

## 3.1 全体の構造

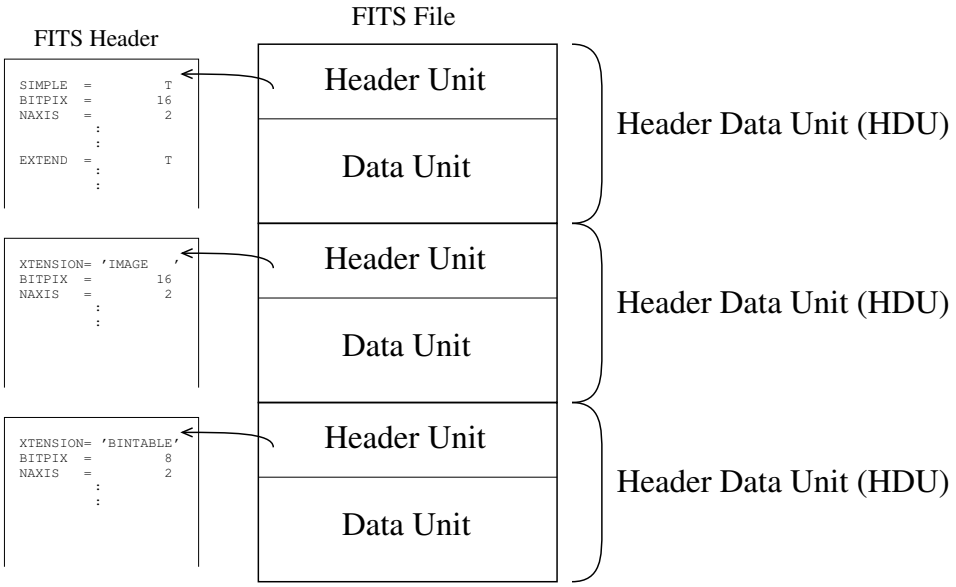



図 2: FITS ファイル全体の構造

FITS ファイルの全体のデータ構造は，図 2のように “Header Unit” と “Data Unit” のペア，すなわち “HDU(Header Data Unit)” を複数個，結合した形をとります．Data Unit は画像やテーブルのようなバイナリデータやアスキーデータを格納するためのもので，Header Unit は「キーワード = 値」の形を羅列した横 80 文字・改行文字なしの plain text で表現され，そこには Data Unit の仕様やデータに関する様々な情報が格納されます．Header Unit，Data Unit それぞれのバイト長は 2880 の整数倍でなければなりません．この規約により生じる各 Unit 最後部の余白については，Header Unit では空白文字 (0x20)，Data Unit では 0x00 で埋められている事が多いようです[13]．

FITS 規約公認の HDU のフォーマットは，“Image”(§3.3)，“ASCII Table”(§3.4) および “Binary Table”(§3.5) の 3 種類です．FITS ファイルにおいて，先頭の HDU は “Primary HDU” と呼ばれ，この HDU については歴史的理由から Image フォーマットでなければならない事になっています．HDU の個数については，Primary HDU のヘッダに記載される `EXTEND` キーワードに対する値が `F` の場合は 1 個である事がわかります．しかし，`EXTEND` が `T` の場合は FITS ファイルを最後までパースしなければ HDU の個数を知る事はできません．

Data Unit のバイナリデータについては，整数値・浮動小数点値 (IEEE 754) ともビッグエンディアンで格納されます．なお，アライメントは一切定められていないので，Data Unit の中身をまるごとメモリにコピーしてエンディアンを変換しても，HDU と処理系によってはメモリアクセスがうまくいかない事があるので注意が必要です．

## 3.2 Header Unit

FITS ファイルの Header Unit の内容については §3.3から具体的に示しますが，今となってはかなり古風な仕様で，Windows の ini ファイルのような値を区分する仕組みもなく，先頭のレコードか

[13] `EXTEND` が `F` の場合については，バイト長を 2880 の整数倍にするための余白が Data Unit にみられない FITS がかなり存在します．SFITSIO ではこれをエラーとはしませんが，規約として正しいのかどうかは著者は知りません．

ら「キーワード = 値 / コメント」の形が始まり，END キーワードで終了するというものです．1 レコードは，印刷可能な 80 文字であり，次の 4 種類のレコード形式があります．

(1) 「キーワード = 値 / コメント」の形式
(2) 「キーワード ␣ 任意の文字列」の形式
(3) 「␣␣␣␣␣␣␣␣ 任意の文字列」の形式
(4) 「END」キーワード

これらのレコードの順番は FITS ファイルの読み書きにより変更しない事が推奨されてはいますが，変わらない保証はありません．また，キーワードは 8 文字までに制限されており，使用が許されているキャラクタは，大文字，数字，「-」および「_」だけです．

(1) の場合は「=」が，(2) の場合は空白文字が 9 文字目に置かれ，9 文字目が英数字などの場合についての規約は存在しません[14]．値は，整数，実数，論理値 (T または F)，文字列の場合があり，文字列の場合のみシングルクォーテーションで囲みます．なお，「/」以降については省略が許されています．

(2) の場合は FITS 規約では，「COMMENT」「HISTORY」しか認められていませんが，実際には CONTINUE や HIERARCH などの規約外キーワードが使われており，キーワードだけで (1)，(2) のどちらの形式かを判定する事はできません．

現在の FITS 規約では，非常に長い文字列値の保存方法は定められていませんが，CONTINUE キーワードを使うコンベンションが広く使われています (§9.1を参照)．

### 3.3 Image HDU

Image HDU は $n$ 次元の画像データを単純な配列として Data Unit に押し込める規格です．データ型は，8-bit 符号無し整数，16-bit・32-bit・64-bit 符号付き整数，32-bit・64-bit 浮動小数点数が使えます．ただし，1 つの Data Unit に複数のデータ型を混在させる事はできません[15]．

Primary HDU または 2 番目以降の HDU に置く事ができますが，FITS のヘッダの最初の部分が Primary とそうでない場合とで異なります．それぞれの場合の例を示します．

```
SIMPLE  =                    T / conformity to FITS standard
BITPIX  =                   32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 1024 / length of data axis 1
NAXIS2  =                 1024 / length of data axis 2
EXTEND  =                    T / possibility of presence of extensions
```

```
XTENSION= 'IMAGE   '           / type of extension
BITPIX  =                   32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                 1024 / length of data axis 1
NAXIS2  =                 1024 / length of data axis 2
PCOUNT  =                    0 / number of parameters per group
GCOUNT  =                    1 / number of groups
```

2 番目以降の HDU では，必ず XTENSION をヘッダの先頭に置き，その値で HDU のタイプを示します．

[14] これはすなわち，過去のデータとの互換を保ちつつキーワードを 9 文字以上に拡張できる余地があるという事です．
[15] Data Unit を malloc() で確保した領域にまるごとコピーし，エンディアンを変換すれば，アライメントの問題なくメモリアクセスが可能という事になります．

データ型は BITPIX の値 (ビットの数) で示し，これが負の場合は浮動小数点型を意味します．データ配列の大きさは，NAXIS$n$ で示し，任意の次元のデータ配列を格納する事ができます．2 次元のデータが最も一般的で，3 次元や 4 次元のデータもたまに見かけます．3 次元の場合は，R,G,B のデータを順に格納する事もあります．これは FITS の規約で決まっているわけではありませんが,「RGB Fits Cube」などと呼ばれ，ds9 等で R,G,B のデータとして扱う事ができます.

読み取るべきピクセルの物理値は，ヘッダの BZERO，BSCALE の値を使って，

$$\text{物理値} = \text{BZERO の値} + \text{データ配列の値} \times \text{BSCALE の値}$$

で決まります．BZERO，BSCALE が存在しない場合は，それぞれ 0.0，1.0 とみなします．BZERO に 32768 をセットし，16-bit 符号無し整数を BITPIX=16 の Data Unit に格納する事はよく行なわれています.

データ型が整数の場合，ヘッダの BLANK で，未定義ピクセルとなるべきデータ配列の値[16] を決める事ができます．データ型が浮動小数点の場合は，NaN が未定義ピクセルとして扱われます.

2 次元の画像データをプロットする場合の原点と座標軸については，FITS の規約で決まっているわけではありませんが，左下を原点とし，右方向，上方向がそれぞれ 1 次元目，2 次元目の座標軸の方向とする事が当然とされています.

WCS については，CTYPE$n$，CRPIX$n$，CRVAL$n$ といったキーワードで，投影方式，参照点等をヘッダ上に記載する事になっています．FITS の規約ではごく基本的なキーワードだけしか定められておらず，より広い範囲をカバーする WCS のコンベンションについては，いわゆる WCS Paper シリーズによって提案されるものが事実上の標準となっています[17]．詳細は，原論文や国立天文台発行の FITS の手引きを参照してください.

WCS を扱う場合，規約上は次の取り決めが存在しない事に注意しなければなりません:

- 左下原点のピクセル座標は，1 なのか，あるいは 0 なのか
- ピクセル座標の整数値が示す位置は，ピクセルの中心なのか左端なのか

一般的には，FITS の左下原点のピクセル座標は 1 から始まり，ピクセル座標の整数値が示す位置はピクセルの中心とされています．ds9 も fv もこれに従っており，たいていの場合はこれ以外の定義を使うメリットは無いと考えられます．ただし，FITS ファイルの作者がこれとは異なる定義を主張する事もありうるので，注意が必要です.

## 3.4 ASCII Table HDU

ASCII Table HDU は，印刷可能な文字からなる 1 つの単純なテーブルを格納するための規格で，Data Unit にはテーブルの内容を格納します．Data Unit のバイトデータは，最初のカラム・最初の行から「行単位で」順に隙間なく格納されます．これはつまり，バイトデータは表の最初のカラムの最初の行から始まり，続いて次のカラムの最初の行，続いてその次のカラムの最初の行…という具合に隙間なく並んでいるという事を意味します.

Header Unit の内容は，Image HDU における BITPIX=8 の 2 次元データの場合の必須キーワードに，テーブルを表現するためのキーワードを追加した形をとっており，Data Unit を単に読み飛ばしたい場合は，ASCII Table HDU であるという事を意識しなくても済むようになっています．ASCII Table HDU は，この後に述べる Binary Table HDU と同様，Primary HDU にはなれません．したがって，次のようにヘッダは必ず XTENSION から始まります.

---

[16] BZERO，BSCALE の値を加味していない，データ配列そのままの値です.

[17] WCS Paper シリーズとは全く別のコンベンションも存在します．代表的なものに DSS の FITS があります．このような特殊な WCS コンベンションも，WCSTools 等の有名なソフトウェアでは扱えるようになっています.

```
XTENSION= 'TABLE   '           / type of extension
BITPIX  =                    8 / number of bits per data element
NAXIS   =                    2 / number of data axes
NAXIS1  =                   21 / width of table in bytes
NAXIS2  =                  256 / number of rows in table
PCOUNT  =                    0 / number of parameters per group
GCOUNT  =                    1 / number of groups
TFIELDS =                    2 / number of fields in each row
TTYPE1  = 'Freq    '           / field name
TBCOL1  =                    1 / starting position in bytes
TFORM1  = 'I10     '           / display format
TTYPE2  = 'L10Pointer'         / field name
TBCOL2  =                   12 / starting position in bytes
TFORM2  = 'I10     '           / display format
EXTNAME = 'HTMLevel10Idx'      / name of this HDU
END
```

XTENSION から TFIELDS まではキーワードの順番が定められており，これらのうち BITPIX，PCOUNT，GCOUNT は，それぞれ 8，0，1 の固定値をとります．

ASCII テーブルのヘッダキーワードの意味は次のとおりです．なお，FITS においてはテーブルのカラムを「フィールド」と呼びますが，ここでは「カラム」で統一しています．

| キーワード | 意味 |
|---|---|
| NAXIS1 | テーブル全体の横方向のバイト数 |
| NAXIS2 | テーブル全体の縦方向のバイト数 |
| TFIELDS | テーブルのカラムの数 |
| TTYPE$n$ | テーブルカラム $n$ の名前 |
| TUNIT$n$ | テーブルカラム $n$ の値の物理単位 |
| TFORM$n$ | テーブルカラム $n$ のデータフォーマット．ANSI FORTRAN-77 フォーマットで，文字列 (“A$w$”)，10 進整数 (“I$w$”)，実数 (“F$w.d$”, “E$w.d$”, “D$w.d$”) を指定する． |
| TBCOL$n$ | テーブルのある行におけるカラム $n$ のバイト単位でのデータ開始点 (1-indexed) |
| TZERO$n$ | テーブルカラム $n$ での値に適用されるゼロ点 |
| TSCAL$n$ | テーブルカラム $n$ での値に適用されるスケーリングファクター |
| TNULL$n$ | テーブルカラム $n$ で未定義値を表すために使う文字列 |

このうち，TZERO$n$，TSCAL$n$ は Image HDU の BZERO，BSCALE と全く同じ役割をするものですが，あまり使われる事はありません．

ASCII Table HDU は構造が単純で，低レベルな FITS I/O ライブラリでも比較的苦労なく作れます．ASCII Table の欠点として，1 つのカラムのデータ幅が 1 つのヘッダレコードから取り出せないという問題があり，それゆえ FITS テンプレート (§8) との相性が良くありません．

### 3.5 Binary Table HDU

Binary Table HDU の規約は，基本的には前述の ASCII Table HDU のバイナリ版とみる事ができますが，きわめて多くのデータ型と様々な拡張機能を含んでおり，とても「SIMPLE」などとは呼べない複雑な代物です．しかし，複雑にした分だけの柔軟性はあり，天文衛星の複雑怪奇なテレメトリデータを格納するような，非常に高度な要求にも応える事ができるようになっています．

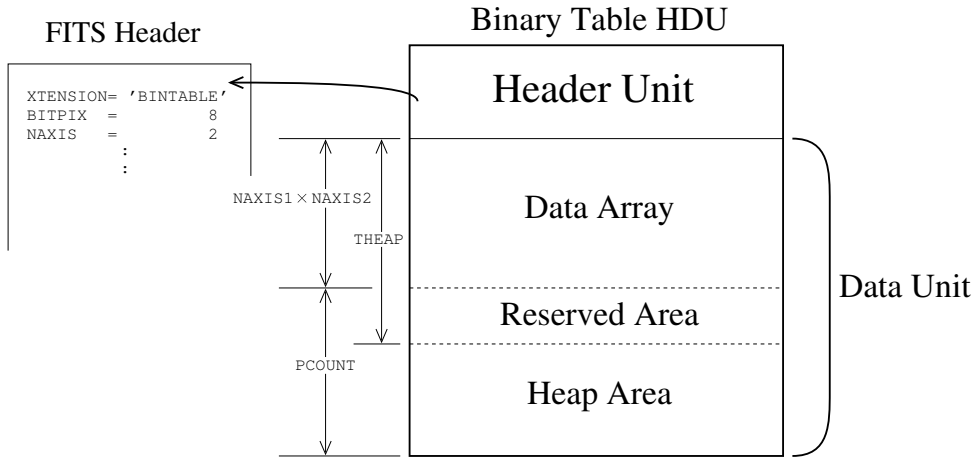



図 3: Binary Table HDU の構造．Data Unit の各領域の大きさは，ヘッダの NAXIS1，NAXIS2，PCOUNT，THEAP の値から得る事ができる．Reserved Area が無い場合，NAXIS1 × NAXIS2 は THEAP と等しく，PCOUNT は Heap Area の大きさとなる．

Binary Table HDU が Image HDU や ASCII Table HDU と最も異なるのは，図 3に示すように Data Unit にはデータ配列 (Data Array) に加えて，予約領域 (Reserved Area) とヒープ領域 (Heap Area) が存在するという点です．データ配列へのテーブルの格納方法は ASCII Table HDU と同様の「行単位」で，隙間無くバイナリデータが格納されます．予約領域はディスクベースの FITS アプリケーションのために用意されているようで，データを格納する領域ではありません[18]．ヒープ領域には，いわゆる「可変長配列」の実データが格納されます．Data Unit を読み飛ばすには，NAXIS1 × NAXIS2 + PCOUNT 分のデータ長に加え，Data Unit が 2880 バイトの整数倍となるための余白分を考慮する必要があります．

次にヘッダの例を示します:

```
XTENSION= 'BINTABLE'           / type of extension
BITPIX  =                    8 / number of bits per data element
NAXIS   =                    2 / number of data axes
NAXIS1  =                   22 / width of table in bytes
NAXIS2  =                 4096 / number of rows in table
PCOUNT  =             12897220 / length of reserved area and heap
GCOUNT  =                    1 / number of groups
TFIELDS =                    6 / number of fields in each row
TTYPE1  = 'ENERG_LO'           / field name
TFORM1  = 'E       '           / data format : 4-byte REAL
TUNIT1  = 'keV     '           / physical unit
(省略)
TTYPE5  = 'N_CHAN  '           / field name
TFORM5  = '1I      '           / data format : 2-byte INTEGER
TTYPE6  = 'MATRIX  '           / field name
TFORM6  = '1PE(3353)'          / data format : variable length of 4-byte REAL
EXTNAME = 'MATRIX  '           / name of this HDU
```

Binary Table HDU は ASCII Table HDU と同様，Primary HDU にはなれず，ヘッダは必ず XTENSION から始まり，TFIELDS まではキーワードの順番が定められています．BITPIX と GCOUNT は固定値でそれぞれ 8，1 をとります．この例では PCOUNT がゼロではなく，THEAP が定義されていないので，予

[18] 筆者はこの予約領域が残された FITS ファイルを見かけた事がありません．

約領域は無くヒープ領域が存在する事がわかります.

Binary Table HDU で使われるヘッダのキーワードと意味は次のとおりです. これを示しながら, 世の中の一般のテーブルと比べた場合の特殊性を説明していきます.

<table>
<tr><th>キーワード</th><th>意味</th></tr>
<tr><td>NAXIS1</td><td>テーブル本体の横方向のバイト数</td></tr>
<tr><td>NAXIS2</td><td>テーブル本体の縦方向のバイト数</td></tr>
<tr><td>TFIELDS</td><td>テーブルのカラムの数</td></tr>
<tr><td>TTYPE$n$</td><td>テーブルカラム $n$ の名前</td></tr>
<tr><td>TUNIT$n$</td><td>テーブルカラム $n$ の値の物理単位</td></tr>
<tr><td>TDISP$n$</td><td>テーブルカラム $n$ の表示フォーマット. FORTRAN-77 形式のフォーマットで, 文字列 (“A$w$”), 論理値 (“L$w$”), 10 進整数 (“I$w$”), 実数 (“F$w.d$”, “E$w.d$”, “G$w.d$”, “D$w.d$”), 2 進整数 (“B$w$”), 8 進整数 (“O$w$”), 16 進整数 (“Z$w$”) が指定可能.</td></tr>
<tr><td>TFORM$n$</td><td>テーブルカラム $n$ のデータ型.
<table>
<tr><th>TFORM$n$ 値</th><th>型</th><th>カラムのバイト長</th></tr>
<tr><td>$r$L</td><td>論理値 (’F’ または’T’)</td><td>$1 \times r$</td></tr>
<tr><td>$r$X</td><td>ビット</td><td>$\lfloor (r-1)/8 \rfloor + 1$</td></tr>
<tr><td>$r$B</td><td>8-bit 符号無し整数</td><td>$1 \times r$</td></tr>
<tr><td>$r$I</td><td>16-bit 符号付き整数</td><td>$2 \times r$</td></tr>
<tr><td>$r$J</td><td>32-bit 符号付き整数</td><td>$4 \times r$</td></tr>
<tr><td>$r$K</td><td>64-bit 符号付き整数</td><td>$8 \times r$</td></tr>
<tr><td>$r$A$a$</td><td>文字 (列)</td><td>$r$</td></tr>
<tr><td>$r$E</td><td>単精度浮動小数点数</td><td>$4 \times r$</td></tr>
<tr><td>$r$D</td><td>倍精度浮動小数点数</td><td>$8 \times r$</td></tr>
<tr><td>$r$C</td><td>単精度複素数</td><td>$4 \times 2 \times r$</td></tr>
<tr><td>$r$M</td><td>倍精度複素数</td><td>$8 \times 2 \times r$</td></tr>
<tr><td>$r$P$t$($e_{\max}$)</td><td>32-bit 配列記述子 (配列長, ヒープオフセット)</td><td>$4 \times 2 \times r$</td></tr>
<tr><td>$r$Q$t$($e_{\max}$)</td><td>64-bit 配列記述子 (配列長, ヒープオフセット)</td><td>$8 \times 2 \times r$</td></tr>
</table>
$r$ はセルの個数で, 1 の場合は省略可能. $a$ は文字列配列を定義する場合の, 1 要素あたりの文字列長 (§9.2を参照). $t$ はヒープ領域を参照するのに使う型. $e_{\max}$ は, 行あたりにヒープ領域を参照する個数の最大値.</td></tr>
<tr><td>TDIM$n$</td><td>テーブルカラム $n$ の多次元配列の定義で, TFORM$n$ の $r$ とは異なり, カラムのバイト長に影響を与えるものではなく, データの解釈として利用される. $r$ が 2 以上の場合に指定でき, ($l$,$m$) の形をとる.</td></tr>
<tr><td>TNULL$n$</td><td>テーブルカラム $n$ が整数型の場合に未定義値を表すために使う値</td></tr>
<tr><td>TZERO$n$</td><td>テーブルカラム $n$ での値に適用されるゼロ点</td></tr>
<tr><td>TSCAL$n$</td><td>テーブルカラム $n$ での値に適用されるスケーリングファクター</td></tr>
</table>

Binary Table HDU においては, TFORM$n$ でのデータ型の理解が最も重要です. このデータ型の定義において最も特徴的なのは, **1 つのカラムに複数のセルを持たせる事ができる**事で, TFORM$n$ の値において $r$ が 2 以上の場合は固定長配列と言います. 例えば TFORM99 = ’48I’ と定義すると, 1 つのカラムに 16-bit 整数が 48 個入る事になります. この様子は, テーブルを平面で捉えるとわかりずらいものですが, テーブルに奥行きがあって, 48 個のセルが奥に向かってずらりと並んでいると考えると理解しやすいかもしれません. さらに加えて TDIM99 = ’(8,6)’ と定義すれば, 48 個のセルを $8 \times 6$ の 2 次元配列と解釈する事になっています. これを多次元配列と呼んでいます.

データ型の定義で最もわかりずらいのが，いわゆる “可変長配列” と呼ばれる TFORM$n$ が $r$P$t$ または $r$Q$t$ の場合です[19]．この場合，セルが奥に向かってずらりと並んでおり，その個数は固定では無いと考えれば良いのですが，データの格納方法が極めて特殊です．可変長配列の場合は，配列データの実体はヒープ領域のどこかに格納され，テーブル本体 (Data Unit のデータ配列) のセルにはその行における “配列長” と “ヒープ領域中の位置 (オフセット)” が格納されています．少し前のページで出てきたヘッダの例で，TFORM6 = ’1PE(3353)’ という定義がありましたが，これは配列の個数の最大は 3353 であり，ヒープ領域のどこかに 32-bit 浮動小数点数 (シンボルは “E”) を格納しているという意味です．例えば，テーブル本体の当該カラムのある行に，(12,34) という数値 (配列記述子) が格納されているとすると，ヒープ領域の先頭から 34 バイトのオフセットの位置から 12 個の 32-bit 浮動小数点数が格納されている事になります．注意しなければならないのは，メインテーブルのカラム・行を問わず，**複数の配列記述子から同一のヒープ領域アドレスへの参照が認められている**という事です[20]．これはすなわち，ヒープ領域すべてをメモリにコピーした場合，一括でエンディアンの変換ができないという事を意味します．もちろん，アライメントの規定もありませんから，メモリ上での読み書きには注意が必要です．さらに，FITS ライブラリにおいて高レベル API を定義する場合，自動化処理のためにはヒープ領域の参照方法に何らかの正規化を考える必要があります．

**NULL 値の定義**は，整数型の場合はヘッダの TNULL$n$ に与えられた値としますが，実数型の場合は NaN を，論理型の場合は文字コード 0 を NULL 値として扱います．文字列型の NULL 値はテーブルの作者が決めます．

TDISP$n$ については，規約上は上記の表にあるものよりも詳細なフォーマット記述子が使えますが，すべてをサポートしているツールやライブラリは少ないようです．

TZERO$n$，TSCAL$n$ は Image HDU の BZERO，BSCALE と全く同じ役割をするもので，画像の場合と同様，符号無し整数値を格納するのにしばしば使われます．

Binary Table HDU のヘッダにおいては，カラムの情報を示すためのキーワードについてのローカル拡張がしばしば行なわれます．有名なものでは，TLMIN$n$，TLMAX$n$，TDMIN$n$，TDMAX$n$ があり，SFITSIO においても，ISAS/JAXA の「あかり」プロジェクトで提案された TALAS$n$(カラムの別名定義)，TELEM$n$(固定長配列での要素名の定義) をサポートしています (§10.8，§10.9)．しかし，このようなローカル拡張のキーワードは，ライブラリによってはカラムの情報なのかどうかを判定できず，カラムの削除や移動などを行なうとヘッダが意図した結果にならないという問題がありますが，残念ながら FITS 標準ではこれに対応した取り決めはありません．SFITSIO では，ISAS/JAXA の L1TSD プロジェクトで考案された TXFLDKWD というキーワードの値に，カラムに関するローカル拡張のキーワード名を羅列してこの問題を解決する方法を提案しています (§10.7)．

---

[19] $r$P$t$ または $r$Q$t$ の $r$ が 2 以上の場合は，1 つのカラムに可変長配列が複数存在する事になりますが，そのような定義を持つ FITS ファイルは見かけた事がありません．ツール類も対応していないものが多いようです．

[20] 同じデータの並びが繰り返されるような場合はヒープ領域を小さくできるという利点はありますが，オンメモリにしろディスクベースにしろ，実装上の困難が多すぎるのも事実です．

# 4 SFITSIO のクラス構造による FITS データ構造の表現と API

§3では，やや詳しく FITS の構造について復習しました．このマニュアルがそのような FITS の解説文を含んでいるのは，すでに §1.3.3で述べたように「SFITSIO の API は FITS の構造そのもの」だからです．

開発者からみると，そのような API を設計する事は，クラスの構造を設計する事とほぼ等価です．そこで重要なのは，目的となる構造物 (今の場合は FITS) をクラスの構造に，**いかに自然な形で「複写」するか**です．逆に，**ライブラリの利用者にとっては，どのように「複写」されているかを知る事が，API を理解する上での基礎**になります．

ここでは，SFITSIO において，FITS の構造を C++のクラスの構造に「複写」している様子と，それらに対する API についてざっと見ていく事にします．このマニュアルはそういった事を知らなくても使えるように作成してはいますが，ある程度は知っておいた方が API の学習もスムーズに行なえると思います．

## 4.1 クラスの構造の全体像と API による HDU へのアクセス

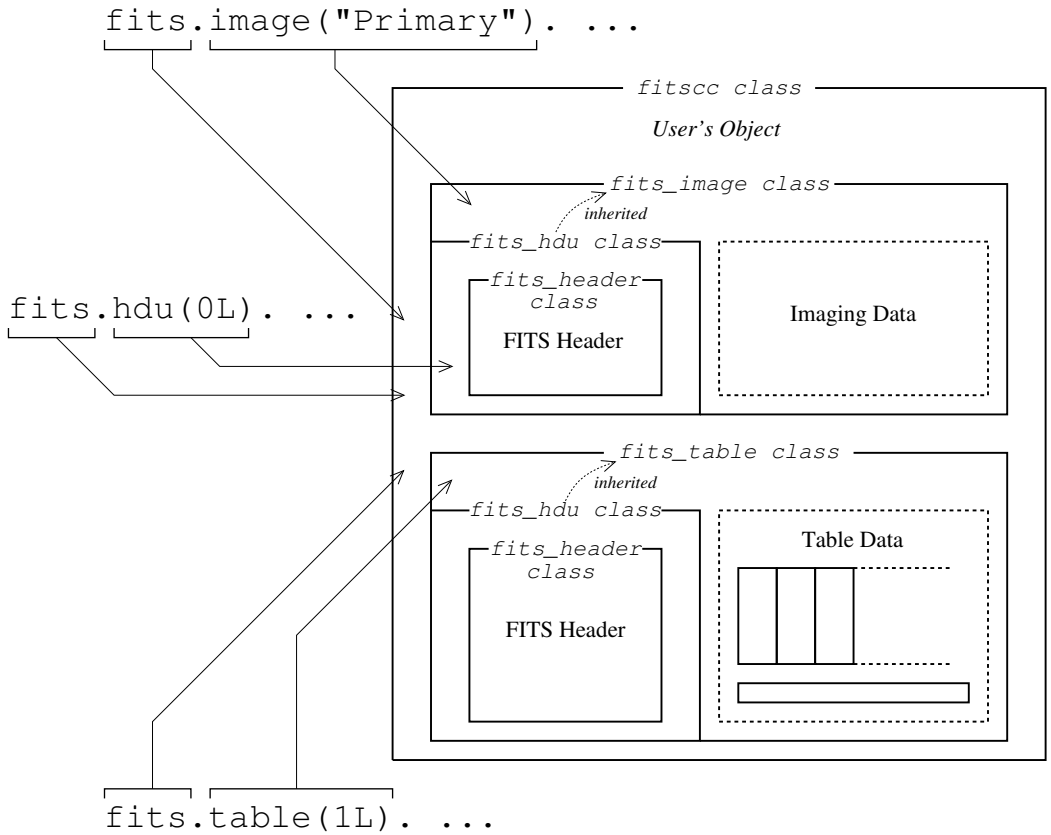



図 4: 1 つの FITS ファイルに相当する「fitscc クラス」のオブジェクトが作られた時の内部構造の概略 (右) と，各 HDU にアクセスするための API(左)．この例では，1 つの FITS ファイルが Image HDU と Binary Table HDU を 1 つずつ持っている場合を示す．

SFITSIO は，FITS ファイルを読む場合も，新規 FITS を作成する場合も，FITS の内容をメモリバッファ上に展開します．このメモリバッファを管理しているのが，最上位クラスである `fitscc` クラスのオブジェクトです．図 4は，`fitscc` クラスのオブジェクトにより，1 つ FITS ファイルの内容

を保持している様子の一例を示したものです．FITS の構造を図示した図 2や図 3と見比べてみてください．FITS のデータ構造が，オブジェクトに「そっくり複写」されている事がおわかりいただけると思います．

図 4から明らかなように，1 つの FITS ファイルを表現するものが `fitscc` クラスで，一般の HDU を表現するものが `fits_hdu` クラスです．Image HDU は `fits_image` クラスとして，ASCII Table HDU または Binary Table HDU は `fits_table` クラスとして表現し，この 2 つのクラスは `fits_hdu` クラスを継承しています．

図 4の左側には，この内部構造に対してどのように API で各 HDU にアクセスするかを示したコードが書かれています．図中の矢印は，API の構造とクラスの構造との対応を示しています．それぞれの HDU へアクセスするには，HDU のタイプがわかっている場合には「`fits.image(...). ...`」，「`fits.table(...). ...`」と書きますが，そうでない場合には「`fits.hdu(...). ...`」と書きます．言うまでもなく，後者の場合は HDU のタイプに依存する情報にはアクセスできません．`hdu(...)` 等の引数は，HDU の通し番号 (0-indexed) か，HDU 名 (ヘッダの `EXTNAME` キーワードの値) を与えます．

## 4.2 FITS ヘッダのクラスの構造と，API によるヘッダへのアクセス

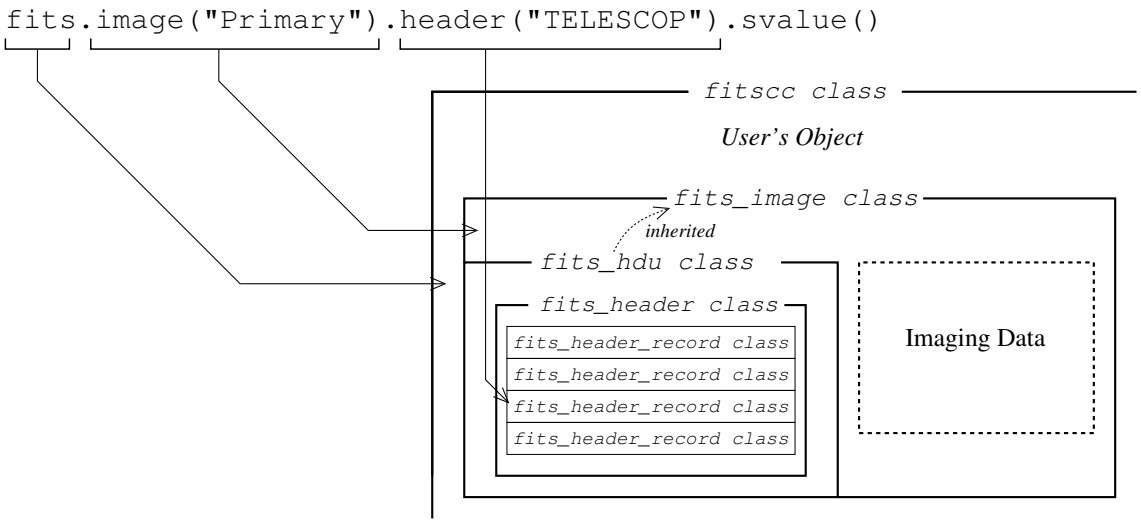



図 5: 図 4の Primary HDU のヘッダ部の詳細なクラス構造と，ヘッダレコードにアクセスするための API の例．

次に，FITS ヘッダまわりの構造をみていきましょう．図 5に示すように，`fits_hdu` クラスは，FITS ヘッダ全体を表現する `fits_header` クラスのオブジェクトを管理下に置いています．さらに，`fits_header` クラスは，1 つのヘッダレコードを表現する `fits_header_record` クラスのオブジェクトの配列を管理下に置いています．

ヘッダの内容にアクセスするための API は，`.image(...)`，`.table(...)`，`.hdu(...)` のいずれかに続いて `.header(...). ...` と書きます．`.header(...)` の引数は，ヘッダレコードの番号 (0-indexed) か，ヘッダのキーワードを与えます．図 5により，クラスの構造との対応がよくわかると思います[21)]．

[21)] 図をよく観察すると，クラスの場合は `fitscc` から `fits_header_record` までの 4 階層に対し，API は `fits` から `header(...)` までの 3 階層である事に気づくと思います．API の方が一階層少ないのはコードを短かく書けるようにするためですが，同一の階層数でないと気持ち悪いという方は，`fits.image(...).header().at(...).svalue()` と書く事もできます．この場合，引数無しの `.header()` は `fits_header` クラスのオブジェクトの参照を返します．

## 4.3 Image HDU の表現

Image HDU は fits_image クラスで表現され，その構造は，ヘッダ関連の内部オブジェクトに加え，画像データを扱うための mdarray クラスの内部オブジェクトが存在するだけです．mdarray は $n$ 次元の配列を扱うための SLLIB が提供する汎用的なクラスで，FITS の Data Unit の画像データをそのまま[22] メモリバッファに持つという単純なものです．

画像データを読み書きしたり編集するには，fits_image クラスのメンバ関数に加え，mdarray クラスのメンバ関数が使えます．前者は，FITS に特化した処理のためのもの (ピクセルへのアクセスから天体画像の一次処理支援まで)，後者は，多次元配列に対する汎用的な処理 (配列に対する数学関数を使った演算等) が行なえるものを揃えてあります．

fits_image クラスのメンバ関数 dvalue(x,y,z)，assign(val,x,y,z) 等を使ってピクセル値にアクセスする例を示します．このコードでは，画像データの (x,y) の値を (x+1,y) にコピーしています．

```
double value = fits.image(0L).dvalue(x, y);
fits.image(0L).assign(value, x+1, y);
```

より高いレベルの API を使った画像解析については §1.1の課題 2，課題 3 でも取り上げました．

mdarray クラスのメンバ関数は,「image(...).data_array().crop(...)」のように，fits_image クラスの data_array() メンバ関数経由で使います．

## 4.4 ASCII Table HDU，Binary Table HDU の表現

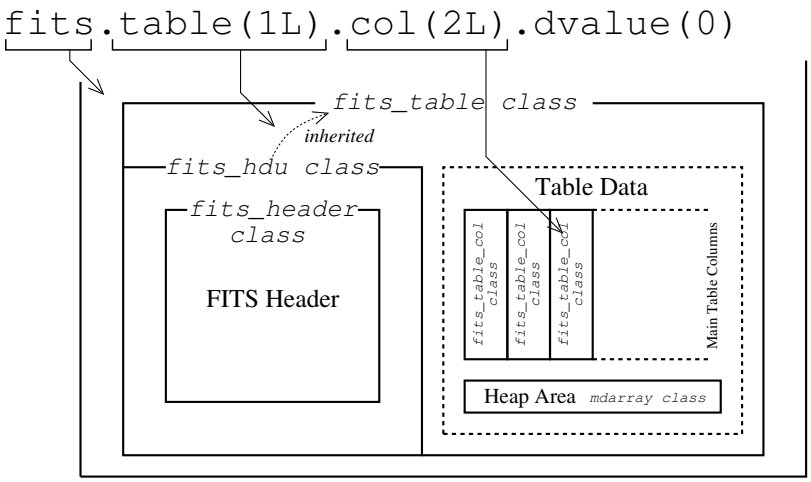



図 6: 図 4の Binary Table HDU の詳細なクラス構造と，バイナリテーブルのセルにアクセスするための API の例．

SFITSIO では，ASCII Table HDU も Binary Table HDU も fits_table クラスで表現します．fits_table クラスは FITS ヘッダを表現する内部オブジェクトに加え，1 つのカラム全体を表現する fits_table_col クラスのオブジェクト配列と，ヒープ領域のための mdarray クラスのオブジェクトを持っています．アライメントを気にせず効率的なデータアクセスができるよう，メインテーブルの内容は**カラムごと**に fits_table_col クラスのオブジェクトが持つバッファに置いています．図

[22] ただし，エンディアンだけは変換しています

6は，クラスの構造と API の構造との対応を示しており，テーブルのカラムに対しては `table(...)` に続いて `.col(...)`．... でアクセスします．

ヒープ領域については，Image HDU の画像バッファと同様，`mdarray` クラスの内部オブジェクトが存在します．可変長配列についての API は，現在は超低レベルのものしかありません．将来的には，`table(...).col(...).dvalue(...)` 等の高レベル API で可変長配列にもアクセスできるようにする予定です．

## 4.5 オブジェクト内の FITS ヘッダ管理の仕組みと注意点

現在の SFITSIO では，Data Unit の内容や構成を変更する場合，プログラマはヘッダの内容を書き換えるメンバ関数を使うのではなく，Data Unit を表現しているクラス (`fits_image`，`fits_table` 等) のメンバ関数を使わなければなりません．これはつまり，プログラマが `fits_header_record` クラスのメンバ関数を使って `BITPIX` のような値を書き込めたとしても，それは FITS ファイルには反映されないという事です．このような書き込みを許可してしまうと，プログラマの混乱を引き起こす恐れがあるため，fits_hdu オブジェクトの管理下にある fits_header オブジェクトでは，特定のキーワードについては自由な書き込みを制限しています[23)]．

次の表に，ユーザプログラムによる自由な書き込みが禁じられているヘッダレコードのキーワード一覧を示します．同時に，各レコード情報に準じた値を書き込むためのメンバ関数を示します．

[23)] 原理的には，ヘッダ関連のメンバ関数が呼ばれてオブジェクト内のヘッダの内容が変更された事を検知し，ライブラリ側で保持している画像やテーブルの管理情報もそれに追従して自動更新する事も可能ですが，SFITSIO はまだそこまで実装ができていません．

| キーワード | 書き込み用メンバ関数 | 章 |
|---|---|---|
| SIMPLE | – | – |
| EXTEND | – | – |
| FMTTYPE | assign_fmttype() | §12.3.21 |
| FTYPEVER | assign_ftypever() | §12.3.22 |
| EXTNAME | assign_hduname() | §12.3.23 |
| EXTVER | assign_hduver() | §12.3.24 |
| BITPIX | image().convert_type() | §12.6.13 |
| NAXIS | image().increase_dim(), etc. | §12.6.20 |
| NAXIS$n$ | image().resize() | §12.6.22 |
| BSCALE | image().assign_bscale() | §12.6.15 |
| BZERO | image().assign_bzero() | §12.6.14 |
| BUNIT | image().assign_bunit() | §12.6.17 |
| BLANK | image().assign_blank() | §12.6.16 |
| XTENSION | table().ascii_to_binary(), etc. | §12.8.36 |
| PCOUNT | – | – |
| GCOUNT | – | – |
| THEAP | table().resize_reserved_area() | §12.9.11 |
| TFIELDS | table().append_cols(), etc. | §12.8.46 |
| TXFLDKWD | table().update_col_header(), etc. | §12.8.41 |
| TTYPE$n$ | table().update_col_header(), etc. | §12.8.41 |
| TDISP$n$ | table().update_col_header(), etc. | §12.8.41 |
| TBCOL$n$ | table().define_a_col() | §12.8.38 |
| TFORM$n$ | table().update_col_header(), etc. | §12.8.41 |
| TDIM$n$ | table().update_col_header(), etc. | §12.8.41 |
| TZERO$n$ | table().update_col_header(), etc. | §12.8.41 |
| TSCAL$n$ | table().update_col_header(), etc. | §12.8.41 |
| TUNIT$n$ | table().update_col_header(), etc. | §12.8.41 |
| TNULL$n$ | table().update_col_header(), etc. | §12.8.41 |
| TELEM$n$ | table().update_col_header(), etc. | §12.8.41 |
| TALAS$n$ | table().update_col_header(), etc. | §12.8.41 |

## 4.6 メンバ関数仕様の策定方針

クラスの構造設計と同様に重要なのが，メンバ関数の名前や API の階層をどのように策定するかです．メンバ関数の具体的な内容については §4.7 で目的ごとに一覧にする事により，全貌をつかみやすくしてありますが，その前に SFITSIO のメンバ関数の策定方針について簡単に解説します．

### 4.6.1 API レベル

SFITSIO では，Header Unit または Data Unit の値の読み書きを行なうメンバ関数については，「FITS 規約に従った演算処理 (BZERO 等) や変換処理を行なう程度」を示す指標として,「高レベル」「低レベル」「超低レベル」の 3 段階に区分しています．これは，高いレベルほど演算処理や変換処理が多くなる事を意味します．

### 4.6.2 引数・返り値のルール

- 共通のルール

  (1) 「長さ」「大きさ」は必ず long 型.
  (2) 文字列型は「`const char *`」か「`char *`」が基本.
  (3) HDU，ヘッダ，画像，テーブル等の要素を示す番号および次元番号は，すべて 0-indexed.
  (4) 動的に作られたオブジェクトのアドレスを返すメンバ関数は存在しない.

- 引数についてのルール

  (1) `const` 属性を持つクラスは，必ず参照.
  (2) x,y,z 等を指定する場合は，次元が小さいものが先.

- 返り値のルール

  (1) 番号を返す関数とステータスを返す関数では，負の値がエラーを示す.
  (2) ファイルの入出力を伴う関数では，返り値はステータス.
  (3) (2) 以外でオブジェクトの内容を改変するメンバ関数 (つまり `const` 属性がつかない関数) については，必ず返り値は自身の参照.

(3) により，例えば次のように画像に対する演算を連続して記述するような事もできます.

```
fits.image(0L).add(image1).add(image2).add(image3). ... ;
```

## 4.7 目的や API レベルごとのメンバ関数一覧

### 4.7.1 ファイル入出力

メンバ関数の返り値はステータスで，失敗した場合は負の値を返します.

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fitscc class**] | | |
| `read_stream(const char *)` | FITS ファイルの読み取り | §12.3.1 |
| `write_stream(const char *)` | FITS ファイルへの書き出し | §12.3.3 |
| `access_stream(const char *)` | コマンド経由での FITS ファイルの読み書き | §12.3.4 |
| `read_template(int, const char *)` | テンプレートファイルの読み取り | §12.3.5 |
| [**fits_header class**] | | |
| `read_stream(cstreamio &)` | Header Unit だけの読み取り | §12.5.1 |
| `write_stream(cstreamio &)` | Header Unit だけの書き出し | §12.5.2 |
| `skip_data_stream(cstreamio &)` | 1 つの Data Unit を読み飛ばし | §12.5.3 |

### 4.7.2 初期化・全内容のコピー

メンバ関数名には必ず「init」がつきます．引数の無い init() は，使用していたメモリ領域を開放する目的にも使えます．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fitscc class**] | | |
| init() | 内容の初期化 | §12.3.15 |
| init(const fitscc &) | 引数のオブジェクトの内容をコピー | §12.3.15 |
| [**fits_hdu class**] | | |
| header_init() | ヘッダの初期化 | §12.4.25 |
| header_init(const fits::header_def []) | ヘッダを初期化し，設定 | §12.4.25 |
| header_init(const fits_header &) | ヘッダの内容をコピー | §12.4.25 |
| [**fits_image class**] | | |
| image(...).init() | 内容の初期化 | §12.6.18 |
| image(...).init(const fits_image &) | 引数のオブジェクトの内容をコピー | §12.6.18 |
| image(...).init(int, long, long, long) | 指定された型とサイズで初期化 | §12.6.18 |
| [**fits_table class**] | | |
| table(...).init() | 内容の初期化 | §12.8.34 |
| table(...).init(const fits_table &) | 引数のオブジェクトの内容をコピー | §12.8.34 |
| table(...).init(const fits::table_def []) | 引数のカラム定義に従って初期化 | §12.8.34 |
| [**fits_table_col class**] | | |
| table(...).col(...).init() | 内容の初期化 | §12.8.35 |
| table(...).col(...).init(const fits_table_col &) | 引数のオブジェクトの内容をコピー | §12.8.35 |

### 4.7.3 全内容の入れ替え

メンバ関数名には必ず「swap」がつきます．ヘッダに関しては，Data Unit の内容に関わる予約キーワードについては入れ替わりません．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_hdu class**] | | |
| header_swap() | 予約キーワード以外について，内容を入れ替え | §12.4.26 |
| [**fits_image class**] | | |
| image(...).swap(fits_image &) | 内容の入れ替え | §12.6.19 |
| [**fits_table class**] | | |
| table(...).swap(fits_table &) | 内容の入れ替え | §12.8.45 |
| [**fits_table_col class**] | | |
| table(...).col(...).swap(fits_table_col &) | 内容の入れ替え | §12.8.60 |

### 4.7.4 データ型を取得

メンバ関数名には必ず「type」がつきます．返り値に関する定数については，§12.1を参照してください．

| メンバ関数 | 返り値 | 詳細 |
|---|---|---|
| [**fits_hdu class**] | | |
| hdu(...).hdutype() | HDU のタイプ | §12.3.13 |
| [**fits_header_record class**] | | |
| hdu(...).header(...).type() | ヘッダレコードのタイプ | §12.4.13 |
| [**fits_image class**] | | |
| image(...).type() | 画像データのデータ型 | §12.6.5 |
| [**fits_table_col class**] | | |
| table(...).col(...).type() | テーブルカラムのデータ型 | §12.8.8 |
| table(...).col(...).heap_is_used() | 可変長配列かどうかを判定 | §12.8.9 |
| table(...).col(...).heap_type() | 可変長配列におけるデータ型 | §12.8.10 |

### 4.7.5 長さ・大きさを取得

メンバ関数名には必ず「bytes」または「length」がつきます．返り値の型はすべて long です．

| メンバ関数 | 返り値 | 詳細 |
|---|---|---|
| [**fitscc class**] | | |
| length() | HDU の個数 | §12.3.7 |
| stream_length() | 出力される非圧縮 FITS ファイルのバイト長 | §12.3.6 |
| [**fits_hdu class**] | | |
| hdu(...).header_length() | ヘッダレコードの個数 | §12.4.1 |
| [**fits_image class**] | | |
| image(...).dim_length() | 次元の数 | §12.6.3 |
| image(...).length() | 全ピクセルの数 | §12.6.4 |
| image(...).length(long) | 引数で指定された次元のピクセルの数 | §12.6.4 |
| image(...).bytes() | 1 ピクセルのバイト長 | §12.6.6 |
| image(...).col_length() | 横方向のピクセルの数 | §12.6.7 |
| image(...).row_length() | 縦方向のピクセルの数 | §12.6.8 |
| image(...).layer_length() | レイヤの数 | §12.6.9 |
| [**fits_table class**] | | |
| table(...).col_length() | カラム (フィールド) の個数 | §12.8.3 |
| table(...).row_length() | 行の個数 | §12.8.4 |
| [**fits_table_col class**] | | |
| table(...).col(...).bytes() | カラムが持つ 1 要素のバイト長 | §12.8.11 |
| table(...).col(...).elem_byte_length() | カラムのバイト長 | §12.8.12 |
| table(...).col(...).elem_length() | カラムが持つ要素数 | §12.8.13 |
| table(...).col(...).dcol_length() | 多次元配列の横方向の要素数 | §12.8.14 |
| table(...).col(...).drow_length() | 多次元配列の縦方向の要素数 | §12.8.15 |
| table(...).col(...).heap_bytes() | 可変長配列の 1 要素のバイト長 | §12.8.16 |
| table(...).col(...).max_array_length() | 可変長配列の長さの最大 | §12.8.17 |
| table(...).col(...).array_length(long) | 引数で指定された行での可変長配列の長さ | §12.8.18 |

### 4.7.6 内部のオブジェクト配列の番号を取得

テーブルのカラム等は，名前でアクセスしても内部の検索機構により高速に動作するようになっていますが，ループの中など呼び出しの回数が多く，パフォーマンスが気になる場合は，番号でアクセスします．そのためには，あらかじめ名前から番号を得ておく必要があり，次に示すメンバ関数を使います．これらは，名前を引数にとり，内部のオブジェクト配列の番号 (long 型で 0-indexed) を返します．関数名には必ず「index」がつきます．指定された引数に該当するものが見つからない場合は負の数を返すので，**存在チェック**にもよく使います．

| メンバ関数 | 返り値 | |
|---|---|---|
| [**fitscc class**] | | |
| index( const char * ) | HDU の番号 | §12.3.14 |
| [**fits_hdu class**] | | |
| hdu(...).header_index( const char * ) | ヘッダレコードの番号 | §12.4.2 |
| [**fits_table class**] | | |
| table(...).col_index( const char * ) | テーブルカラムの番号 | §12.8.6 |

### 4.7.7 構造物の名前やバージョンを取得・設定

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fitscc class**] | | |
| fmttype() | FMTTYPE の値 (文字列) を返す | §12.3.8 |
| assign_fmttype(...) | FMTTYPE の値を設定 | §12.3.21 |
| ftypever() | FTYPEVER の値 (整数) を返す | §12.3.9 |
| assign_ftypever(...) | FTYPEVER の値を設定 | §12.3.22 |
| [**fits_hdu class**] | | |
| hdu(...).hduname() | HDU の名前 (EXTNAME の値; 文字列) を返す | §12.3.10 |
| hdu(...).assign_hduname(...) | HDU の名前 (EXTNAME の値; 文字列) を設定 | §12.3.23 |
| hdu(...).hduver() | HDU のバージョン (EXTVER の値; 整数) を返す | §12.3.11 |
| hdu(...).assign_hduver(...) | HDU のバージョン (EXTVER の値; 整数) を設定 | §12.3.24 |
| hdu(...).hduver_is_set() | HDU のバージョン (EXTVER の値; 整数) の存在をチェック | §12.3.26 |
| hdu(...).hdulevel() | HDU のレベル (EXTLEVEL の値; 整数) を返す | §12.3.12 |
| hdu(...).assign_hdulevel(...) | HDU のレベル (EXTLEVEL の値; 整数) を設定 | §12.3.25 |
| hdu(...).hdulevel_is_set() | HDU のレベル (EXTLEVEL の値; 整数) の存在をチェック | §12.3.27 |
| [**fits_table_col class**] | | |
| table(...).col(...).name() | カラムの名前を返す | §12.8.7 |
| table(...).col(...).assign_name(...) | カラムの名前を設定 | §12.8.37 |

### 4.7.8 データの読み取り (高レベル)

FITS 特有の演算処理を行ない，返すべき型に変換された値を得るためのメンバ関数群で，SFITSIO ではプログラマがデータ読み取りのために通常使用するものと位置付けています．

ヘッダの文字列値については，左右のクォーテーションと空白文字とを除去し，文字列中のクォーテーションを表現する「’’」を「’」に変換したものを返します．

Data Unit から値を取り出す場合は，ヘッダの BZERO，BSCALE，TZERO$n$，TSCAL$n$ の値を加味します．さらに，dvalue()，svalue()，get_svalue() については，データ値がヘッダの BLANK または TNULL$n$ の値に一致する場合，それぞれ NaN，NULL 文字列 (デフォルトでは"NULL")[24] を返します．

- **データを読み出し，返り値として返す**

  メンバ関数名は必ず dvalue()，lvalue()，llvalue()，bvalue() svalue() のいずれかで，それぞれ double 型，long 型，long long 型，bool 型，文字列型 (const char *) を返します．

| メンバ関数 | 返り値の型 | 詳細 |
|---|---|---|
| [**fits_header_record class**] | | |
| hdu(...).header(...).dvalue() | double | §12.4.6 |
| hdu(...).header(...).lvalue() | long | §12.4.7 |
| hdu(...).header(...).llvalue() | long long | §12.4.7 |
| hdu(...).header(...).bvalue() | bool | §12.4.8 |
| hdu(...).header(...).svalue() | const char * | §12.4.4 |
| [**fits_image class**] | | |
| image(...).dvalue(long,long,long) | double | §12.6.10 |
| image(...).lvalue(long,long,long) | long | §12.6.11 |
| image(...).llvalue(long,long,long) | long long | §12.6.11 |
| [**fits_table_col class**] | | |
| table(...).col(...).dvalue(...) | double | §12.8.20 |
| table(...).col(...).lvalue(...) | long | §12.8.21 |
| table(...).col(...).llvalue(...) | long long | §12.8.21 |
| table(...).col(...).bvalue(...) | bool | §12.8.22 |
| table(...).col(...).svalue(...) | const char * | §12.8.23 |

- **データを読み出し，引数のバッファへ保存**

  メンバ関数名は「get_svalue()」です．文字列値を，引数に与えたバッファに格納します．

| メンバ関数 | 詳細 |
|---|---|
| [**fits_header_record class**] | |
| hdu(...).header(...).get_svalue(char *, size_t) | §12.4.5 |
| [**fits_table_col class**] | |
| table(...).col(...).get_svalue(long, char *, size_t) 等 | §12.8.24 |

### 4.7.9 データの書き込み (高レベル)

メンバ関数名は必ず「assign()」で，引数に与えられた変数の型によって，FITS 特有の演算処理と型変換を行ない，FITS のデータバッファへ値を格納するためのメンバ関数群で，SFITSIO ではプログラマがデータ書き込みのために通常使用するものと位置付けています．

ヘッダの文字列値については，左右のクォーテーションを追加し，文字列中のクォーテーション「'」は「''」に変換します．

[24] NULL 文字列は，table(...).assign_null_svalue(...) で変更可能です．詳細は §12.8.29をご覧ください．

Data Unit へ値を書き込む場合は，ヘッダの BZERO，BSCALE，TZERO$n$，TSCAL$n$ の値を加味します．さらに，引数が浮動小数点値または文字列値の場合については，それぞれ NaN，NULL 文字列 (デフォルトでは"NULL")[25] を与えられた場合は，ヘッダの BLANK または TNULL$n$ の値を格納します．

| メンバ関数 | 詳細 |
|---|---|
| [**fits_header_record class**] | |
| `hdu(...).header(...).assign(...)` | §12.4.9 |
| [**fits_image class**] | |
| `image(...).assign(...)` | §12.6.12 |
| [**fits_table_col class**] | |
| `table(...).col(...).assign(...)` | §12.8.25 |

[25] NULL 文字列は，`table(...).assign_null_svalue(...)` で変更可能です．詳細は §12.8.29をご覧ください．

### 4.7.10 データの読み取り (低レベル)

FITS 特有の演算処理を行なわずに，返すべき型に変換された値を得るためのメンバ関数群で，ヘッダの BZERO，BSCALE，BLANK，TZERO$n$，TSCAL$n$，TNULL$n$ の設定が無い場合や，パフォーマンスアップを狙う場合に使用します (とにかくパフォーマンスを得たい場合は，超低レベル API を使う方が良いでしょう)．FITS のデータ型とメンバ関数のデータ型が一致する場合に，最も高速に動作します (ただし，ビット型は演算量がやや多いです)．

ヘッダの文字列値については，左右のクォーテーションの除去を行ないません．Data Unit から値を取り出す場合，保存されているデータ値に対して BZERO，BLANK 等のための演算は行ないません．

- **データを読み出し，返り値として返す**

  ヘッダの場合はメンバ関数名は単に value() で，Data Unit の場合はメンバ関数名は double_value()，long_value() のように型名をフルに表記したものとなっています．

| メンバ関数 | 返り値の型 | 詳細 |
|---|---|---|
| [**fits_header_record class**] | | |
| hdu(...).header(...).value() | const char * | §12.4.17 |
| [**fits_image class**] | | |
| image(...).double_value(long,long,long) | double | §12.7.5 |
| image(...).float_value(long,long,long) | float | §12.7.6 |
| image(...).longlong_value(long,long,long) | long long | §12.7.7 |
| image(...).long_value(long,long,long) | long | §12.7.8 |
| image(...).short_value(long,long,long) | short | §12.7.9 |
| image(...).byte_value(long,long,long) | unsigned char | §12.7.10 |
| [**fits_table_col class**] | | |
| table(...).col(...).double_value(...) | double | §12.9.17 |
| table(...).col(...).float_value(...) | float | §12.9.16 |
| table(...).col(...).longlong_value(...) | long long | §12.9.14 |
| table(...).col(...).long_value(...) | long | §12.9.13 |
| table(...).col(...).short_value(...) | short | §12.9.12 |
| table(...).col(...).byte_value(...) | unsigned char | §12.9.15 |
| table(...).col(...).logical_value(...) | int | §12.9.19 |
| table(...).col(...).bit_value(...) | long | §12.9.18 |
| table(...).col(...).string_value(...) | const char * | §12.9.20 |
| table(...).col(...).array_heap_offset(...) | long | §12.9.21 |

- **データを読み出し，引数のバッファへ保存**

  メンバ関数名はヘッダの場合は「get_value()」，Data Unit の場合は「get_string_value()」です．文字列値を，引数に与えたバッファに格納します．

| メンバ関数 | 詳細 |
|---|---|
| [**fits_header_record class**] | |
| hdu(...).header(...).get_value(char *, size_t) | §12.4.18 |
| [**fits_table_col class**] | |
| table(...).col(...).get_string_value(long, char *, size_t) 等 | §12.9.22 |

### 4.7.11 データの書き込み (低レベル)

FITS 特有の演算処理を行なわずに，FITS のデータ型に変換された値を書き込むためのメンバ関数群で，ヘッダの BZERO，BSCALE，BLANK，TZERO$n$，TSCAL$n$，TNULL$n$ の設定が無い場合や，パフォーマンスアップを狙う場合に使用します (とにかくパフォーマンスを得たい場合は，超低レベル API を使う方が良いでしょう)．FITS のデータ型とメンバ関数のデータ型が一致する場合に，最も高速に動作します (ただし，ビット型は演算量がやや多いです)．

ヘッダの文字列値については，左右のクォーテーションの追加処理等は行ないません．Data Unit へ値を書き込む場合，BZERO，BLANK 等のための演算は行ないません．

| メンバ関数 | 詳細 |
|---|---|
| [**fits_header_record class**] | |
| hdu(...).header(...).assign_value(...) 等 | §12.4.21 |
| [**fits_image class**] | |
| image(...).assign_double(...) | §12.7.11 |
| image(...).assign_float(...) | §12.7.12 |
| image(...).assign_longlong(...) | §12.7.13 |
| image(...).assign_long(...) | §12.7.14 |
| image(...).assign_short(...) | §12.7.15 |
| image(...).assign_byte(...) | §12.7.16 |
| [**fits_table_col class**] | |
| table(...).col(...).assign_double(...) | §12.9.28 |
| table(...).col(...).assign_float(...) | §12.9.27 |
| table(...).col(...).assign_longlong(...) | §12.9.25 |
| table(...).col(...).assign_long(...) | §12.9.24 |
| table(...).col(...).assign_short(...) | §12.9.23 |
| table(...).col(...).assign_byte(...) | §12.9.26 |
| table(...).col(...).assign_logical(...) | §12.9.30 |
| table(...).col(...).assign_bit(...) | §12.9.29 |
| table(...).col(...).assign_string(...) | §12.9.31 |
| table(...).col(...).assign_arrdesc(...) | §12.9.32 |

### 4.7.12 データへのアクセス (超低レベル)

ハイリスク・ハイリターンのメンバ関数群で，Data Unit の生のバイトデータを直接読み書きします．バイナリテーブルのヒープ領域を除き，オブジェクト内のバッファはエンディアンが変換され，アライメントは処理系に適合しているため，正しい型のポインタ変数にアドレスを取得すれば，ただちにデータにアクセスできます．SFITSIO では，FITS ファイル内で使われるデータ型 (`fits::logical_t`, `fits::double_t` など) が定義されていますので，ポインタ変数の宣言にはそれらを使うと良いでしょう (型の定義については §12.2を参照)．

最高のパフォーマンスを狙う場合に使用しますが，`void *`**型でデータをやりとりするものについては，型のチェック等はプログラマに任せられており，よく理解しないで使うと思わぬ結果を招く可能性があります**．

メンバ関数名は `get_data()`，`put_data()`，`data_ptr()`，型名`_ptr()` であり，バイナリテーブルのヒープ領域については，`get_heap()`，`put_heap()`，`heap_ptr()` の名称になっています．なお，ヒープ領域のバッファについては，エンディアンが変換されておらず (原理的に変換不可能)，アライメントは不定です．2 バイト以上の型のデータを読み書きする場合はご注意ください．可変長配列に関する API については，§4.7.21 にまとめてあります．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_image class**] | | |
| `image(...).get_data(...)` | バイトデータを引数のバッファに取得 | §12.7.3 |
| `image(...).put_data(...)` | 引数のバッファのバイトデータを書き込み | §12.7.4 |
| `image(...).byte_t_ptr(...)` | 内部バッファのアドレス (`fits::byte_t *`型) を取得 | §12.7.2 |
| `image(...).short_t_ptr(...)` | 内部バッファのアドレス (`fits::short_t *`型) を取得 | §12.7.2 |
| `image(...).long_t_ptr(...)` | 内部バッファのアドレス (`fits::long_t *`型) を取得 | §12.7.2 |
| `image(...).longlong_t_ptr(...)` | 内部バッファのアドレス (`fits::longlong_t *`型) を取得 | §12.7.2 |
| `image(...).float_t_ptr(...)` | 内部バッファのアドレス (`fits::float_t *`型) を取得 | §12.7.2 |
| `image(...).double_t_ptr(...)` | 内部バッファのアドレス (`fits::double_t *`型) を取得 | §12.7.2 |
| `image(...).data_ptr(...)` | 内部バッファのアドレス (`void *`型) を取得 | §12.7.2 |
| [**fits_table_col class**] | | |
| `table(...).col(...).get_data(...)` | テーブル本体のカラムデータ領域用 | §12.9.3 |
| `table(...).col(...).put_data(...)` | | §12.9.4 |
| `table(...).col(...).bit_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).byte_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).logical_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).ascii_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).short_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).long_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).longlong_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).float_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).double_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).complex_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).doublecomplex_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).longarrdesc_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).llongarrdesc_t_ptr(...)` | | §12.9.2 |
| `table(...).col(...).data_ptr()` | | §12.9.2 |
| [**fits_table class**] | | |
| `table(...).get_heap(...)` | バイナリテーブルのヒープ領域用 | §12.9.6 |
| `table(...).put_heap(...)` | | §12.9.7 |
| `table(...).heap_ptr()` | | §12.9.5 |

### 4.7.13 データ型の変換

数値型のデータを高速変換を行なうメンバ関数です．バイナリテーブルの convert_type() で変換できるのは，可変長配列ではない数値型のカラムのみです．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_image class**] | | |
| image(...).convert_type(...) | 指定された型，ZERO 値，SCALE 値，BLANK 値を持つ画像へ変換する | §12.6.13 |
| [**fits_table_col class**] | | |
| table(...).col(...).convert_type(...) | 指定された型, ZERO 値, SCALE 値, NULL 値を持つカラムへ変換する | §12.8.28 |

### 4.7.14 ZERO 値，SCALE 値，BLANK 値，UNIT 値に関するメンバ関数

バイナリテーブル，ASCII テーブルの TDIM$n$ 等の変更については，§4.7.19をご覧ください．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_image class**] | | |
| image(...).bzero() | BZERO の値を取得 | §12.6.14 |
| image(...).assign_bzero() | BZERO の値を設定 | §12.6.14 |
| image(...).bscale() | BSCALE の値を取得 | §12.6.15 |
| image(...).assign_bscale() | BSCALE の値を設定 | §12.6.15 |
| image(...).blank() | BLANK の値を取得 | §12.6.16 |
| image(...).assign_blank() | BLANK の値を設定 | §12.6.16 |
| image(...).bunit() | BUNIT の値を取得 | §12.6.17 |
| image(...).assign_bunit() | BUNIT の値を設定 | §12.6.17 |
| [**fits_table_col class**] | | |
| table(...).col(...).tzero() | TZERO の値を取得 | §12.8.30 |
| table(...).col(...).assign_tzero() | TZERO の値を設定 | §12.8.30 |
| table(...).col(...).tscal() | TSCAL の値を取得 | §12.8.31 |
| table(...).col(...).assign_tscal() | TSCAL の値を設定 | §12.8.31 |
| table(...).col(...).tnull() | TNULL の値を取得 | §12.8.32 |
| table(...).col(...).assign_tnull() | TNULL の値を設定 | §12.8.32 |
| table(...).col(...).tunit() | TUNIT の値を取得 | §12.8.33 |
| table(...).col(...).assign_tunit() | TUNIT の値を設定 | §12.8.33 |

### 4.7.15 データの編集

バッファ領域サイズの変更，要素の追加・挿入・削除などのための関数です．それぞれ，resize，append，insert，erase で統一しています．クラスによっては，これら以外の名前を持つメンバ関数もあります.

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fitscc class**] | | |
| append_image(...) | Image HDU の追加 | §12.3.16 |
| append_table(...) | ASCII Table または Binary Table HDU の追加 | §12.3.17 |
| insert_image(...) | Image HDU の挿入 | §12.3.18 |
| insert_table(...) | ASCII Table または Binary Table HDU の挿入 | §12.3.19 |
| erase(...) | HDU の削除 | §12.3.20 |
| [**fits_hdu class**] | | |
| hdu(...).header_append_records(...) | 複数のヘッダレコードの追加 | §12.4.27 |
| hdu(...).header_append(...) | 1 つのヘッダレコードの追加 | §12.4.28 |
| hdu(...).header_insert_records(...) | 複数のヘッダレコードの挿入 | §12.4.29 |
| hdu(...).header_insert(...) | 1 つのヘッダレコードの挿入 | §12.4.30 |
| hdu(...).header_erase_records(...) | 複数のヘッダレコードの削除 | §12.4.31 |
| hdu(...).header_erase(...) | 1 つのヘッダレコードの削除 | §12.4.32 |
| hdu(...).header_update(...) | 1 つのヘッダレコードを追加，更新 (キーワードが存在しない場合は追加) | §12.4.23 |
| hdu(...).header_assign(...) | 1 つのヘッダレコードを更新 | §12.4.24 |
| hdu(...).header(...).assign_comment(...) | ヘッダレコードのコメントを設定 | §12.4.22 |
| hdu(...).header_rename(...) | 1 つのヘッダレコードのキーワードを変更 | §12.4.33 |
| [**fits_image class**] | | |
| image(...).increase_dim(...) | 次元を 1 つ増やす | §12.6.20 |
| image(...).decrease_dim(...) | 次元を 1 つ減らす | §12.6.21 |
| image(...).resize(...) | 各次元の長さの変更 | §12.6.22 |
| image(...).assign_default(...) | .resize() 等で要素長を大きくした場合の新規ピクセル値を設定 | §12.6.23 |
| image(...).data_array().crop(...) 等 | mdarray クラスのメンバ関数を使って編集．§4.7.17を参照. | §12.7.1 |

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_table class**] | | |
| table(...).resize_rows(...) | 行の大きさの変更 | §12.8.51 |
| table(...).append_rows(...) | 行の追加 | §12.8.52 |
| table(...).insert_rows(...) | 行の挿入 | §12.8.53 |
| table(...).erase_rows(...) | 行の削除 | §12.8.54 |
| table(...).clean_rows(...) | 行の初期化 | §12.8.55 |
| table(...).move_rows(...) | 行のコピー | §12.8.56 |
| table(...).swap_rows(...) | 行の入れ替え | §12.8.57 |
| table(...).import_rows(...) | テーブルのインポート | §12.8.58 |
| table(...).append_cols(...) | 複数のカラムの追加 | §12.8.46 |
| table(...).append_a_col(...) | 1 つのカラムの追加 | §12.8.46 |
| table(...).insert_cols(...) | 複数のカラムの挿入 | §12.8.47 |
| table(...).insert_a_col(...) | 1 つのカラムの挿入 | §12.8.47 |
| table(...).swap_cols(...) | カラムの入れ替え | §12.8.48 |
| table(...).erase_cols(...) | 複数のカラムの削除 | §12.8.49 |
| table(...).erase_a_col(...) | 1 つのカラムの削除 | §12.8.49 |
| table(...).copy(...) | 別オブジェクトへのコピー | §12.8.50 |
| table(...).assign_null_svalue(...) | 高レベル API における NULL 文字列の設定 | §12.8.29 |
| table(...).ascii_to_binary() | ASCII テーブルをバイナリテーブルに変換 | §12.8.36 |
| [**fits_table_col class**] | | |
| table(...).col(...).move(...) | カラム内での行のコピー | §12.8.59 |
| table(...).col(...).swap(...) | カラム内での行の入れ替え | §12.8.60 |
| table(...).col(...).clean(...) | カラム内での値の初期化 | §12.8.61 |
| table(...).col(...).import(...) | 特定のカラムでのインポート | §12.8.62 |
| table(...).col(...).assign_default(...) | .resize_rows() 等で行の長さを大きくした場合の新規セル値を設定 | §12.8.63 |

### 4.7.16 画像処理のためのメンバ関数

矩形領域のピクセルを対象とする処理は，プログラマのコードでループを組むより，ユーザ関数の呼び出しに対応したメンバ関数を使った方が，簡単に高速動作するものが作れます．ユーザ関数に与えられるピクセル値は，画像のタイプにかかわらず BZERO 値，BSCALE 値，BLANK 値の変換処理済みの浮動小数点型実数ですので，処理内容の本質ではないコードを最小化できます．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_image class**] | | |
| `image(...).fix_rect_args(...)` | `.scan_cols()` 等に与えるための領域指定のための引数の値をチェックし，オブジェクトが持つ画像の範囲内に入るように引数の値を修正 | §12.6.24 |
| `image(...).scan_cols(double [], ...)` | 指定領域をユーザ関数で横方向にスキャン | §12.6.25 |
| `image(...).scan_rows(double [], ...)` | 指定領域をユーザ関数で縦方向にスキャン | §12.6.26 |
| `image(...).scan_layers(double [], ...)` | 指定領域をユーザ関数でレイヤ方向にスキャン | §12.6.27 |
| `image(...).fill(double, ...)` | 指定領域のピクセル値を 1 つの値に変更 | §12.6.28 |
| `image(...).fill(double, void (*)(...), ...)` | 指定領域をユーザ関数の返り値で埋める | §12.6.33 |
| `image(...).add(double, ...)` | 指定領域のピクセル値を 1 つの値で加算 | §12.6.29 |
| `image(...).subtract(double, ...)` | 指定領域のピクセル値を 1 つの値で減算 | §12.6.30 |
| `image(...).multiply(double, ...)` | 指定領域のピクセル値を 1 つの値で乗算 | §12.6.31 |
| `image(...).divide(double, ...)` | 指定領域のピクセル値を 1 つの値で除算 | §12.6.32 |
| `image(...).copy(fits_image *, ...)` | 指定領域を別の `fits_image` オブジェクトにコピー | §12.6.34 |
| `image(...).cut(fits_image *, ...)` | 指定領域を別の `fits_image` オブジェクトにコピーし，コピー元画像の該当領域を消去 | §12.6.34 |
| `image(...).paste(const fits_image &, ...)` | 別の `fits_image` オブジェクトの画像を貼り付け | §12.6.35 |
| `image(...).paste(const fits_image &, void (*)(...), ...)` | 別の `fits_image` オブジェクトの画像をユーザ関数による演算経由で貼り付け | §12.6.40 |
| `image(...).add(const fits_image &, ...)` | 別の `fits_image` オブジェクトの画像を加算 | §12.6.36 |
| `image(...).subtract(const fits_image &, ...)` | 別の `fits_image` オブジェクトの画像で減算 | §12.6.37 |
| `image(...).multiply(const fits_image &, ...)` | 別の `fits_image` オブジェクトの画像を乗算 | §12.6.38 |
| `image(...).divide(const fits_image &, ...)` | 別の `fits_image` オブジェクトの画像で除算 | §12.6.39 |

### 4.7.17 画像の編集に使える mdarray クラスのメンバ関数

FITS の画像データは，fits_image クラスと mdarray クラスのメンバ関数で編集が可能です．ここでは，mdarray クラスのメンバ関数をいくつか紹介します．詳細は，SLLIB のマニュアルかヘッダファイル `mdarray.h` のコメントをご覧ください．

| メンバ関数 | 動作 |
|---|---|
| [**mdarray class**] | |
| `image(...).data_array().swap(...)` | 指定した次元で区間データの入れ替え |
| `image(...).data_array().move(...)` | 指定した次元で区間データのコピー |
| `image(...).data_array().cpy(...)` | 指定した次元で区間データのコピー (バッファのサイズは必要に応じて変更) |
| `image(...).data_array().insert(...)` | 指定した次元でブランクデータを挿入 |
| `image(...).data_array().crop(...)` | 指定した次元で一部区間の切り出し |
| `image(...).data_array().erase(...)` | 指定した次元で一部区間の削除 |
| `image(...).data_array().round()` 等 | 全ピクセルに対して演算し，その結果を格納．`.ceil()`, `.floor()`, `.round()`, `.trunc()`, `.abs()` が利用可能. |

### 4.7.18 画像解析のためのメンバ関数

fits_header_record クラスと fits_image クラスでは，画像の一次リダクションで必須の機能を持つメンバ関数を提供しています．SFITSIO のパッケージに含まれる，`tools/stat_pixels.cc`, `tools/combine_images.cc` も参考にしてください．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_header_record class**] | | |
| `hdu(...).header(...).get_section_info(...)` | FITS ヘッダにおける，IRAF 形式の矩形領域情報 (`BIASSEC` 等) の取得 | §12.4.36 |
| [**fits_image class**] | | |
| `image(...).stat_pixels(...)` | 指定領域の平均値，標準偏差，中央値などの算出 | §12.6.41 |
| `image(...).combine_layers(...)` | 3 次元画像の指定領域を，平均値や中央値等で画像コンバイン | §12.6.42 |

### 4.7.19 テーブルのカラム定義の操作に関するメンバ関数

次に示す「col_header」がついたメンバ関数は，ASCII Table または Binary Table のカラム定義に関する予約キーワード (例: TDISP$n$ 等) と非予約キーワード (例: TLMAX$n$ 等) との両方を扱う事ができます．次の例のように，カラム名 (番号) とキーワード名はそれぞれ別の引数で指定できるため，snprintf() などを使って $n$ を含んだキーワードを作る必要はありません．

```
/* "EVENT" テーブルのカラム "R-BAND" の TUNIT の値とコメントを変更 */
fits.table("EVENT").update_col_header("R-BAND", "TUNIT",
                                      "mag", "absolute magnitude");
```

予約キーワードの値を変更した場合，オブジェクト内部のデータも自動的に更新されます．非予約キーワードの場合は，TXFLDKWD も自動的に更新されます (TXFLDKWD については §10.7を参照)．

define(...) メンバ関数は，構造体 fits::table_def (§12.2) を使って予約キーワードの値を変更します．

| メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_table class**] | | |
| table(...).col_header_index(...) | 指定されたカラム定義のヘッダレコードの番号を返す | §12.8.39 |
| table(...).col_header(...).svalue() 等 | 指定されたカラム定義のヘッダレコードにアクセスする | §12.8.40 |
| table(...).update_col_header(...) | カラム定義の 1 つのヘッダレコードを変更する | §12.8.41 |
| table(...).erase_col_header(...) | カラム定義の 1 つのヘッダレコードを削除する | §12.8.42 |
| table(...).rename_col_header(...) | ユーザ定義のカラム用ヘッダキーワードの名前を変更する | §12.8.43 |
| table(...).sort_col_header() | カラム用ヘッダキーワードをカラム順にソートする | §12.8.44 |
| [**fits_table_col class**] | | |
| table(...).col(...).definition() | 1 つのカラム定義を読み取る | §12.8.19 |
| table(...).col(...).define(...) | 1 つのカラム定義を変更 | §12.8.38 |

### 4.7.20 ヘッダ処理に関する特別なメンバ関数

| 関数・メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_hdu class**] | | |
| hdu(...).header(...).status() | ヘッダレコードの状態を取得 | §12.4.14 |
| hdu(...).header(...).keyword() | ヘッダレコードのキーワードを取得 | §12.4.15 |
| hdu(...).header(...).get_keyword(...) | ヘッダレコードのキーワードを取得 | §12.4.16 |
| hdu(...).header(...).comment() | ヘッダレコードのコメントを取得 | §12.4.19 |
| hdu(...).header(...).get_comment(...) | ヘッダレコードのコメントを取得 | §12.4.20 |
| hdu(...).header_regmatch(...) | 正規表現でキーワードを検索する | §12.4.3 |
| hdu(...).header(...).assign_system_time() | 現在の日付時刻 (UTC) を「yyyy-mm-ddThh:mm:ss」の形式でセットする | §12.4.37 |
| hdu(...).header_formatted_string() | FITS ファイル用にフォーマットされたヘッダ文字列を返す | §12.4.35 |
| hdu(...).header_fill_blank_comments() | コメントが存在しない場合，SFITSIO が持つヘッダコメント辞書の内容で埋める | §12.4.38 |
| hdu(...).header_assign_default_comments() | コメントの存在にかかわらず，SFITSIO が持つヘッダコメント辞書の内容で埋める | §12.4.39 |
| [**関数**] | | |
| fits::update_comment_dictionary(...) | コメント辞書の内容を追加・変更する | §12.4.40 |

### 4.7.21 バイナリテーブルの可変長配列の操作に関するメンバ関数 (低レベル)

現在の SFITSIO では，高レベル API では可変長配列をサポートしないので，下記の低レベル API を使う必要があります．SFITSIO のソースパッケージにあるサンプルコード test/access_bte_heap.cc, sample/create_vl_array.cc もご覧ください．

| 関数・メンバ関数 | 動作 | 詳細 |
|---|---|---|
| [**fits_table class**] | | |
| table(...).get_heap(...) | バイナリテーブルのヒープ領域用 | §12.9.6 |
| table(...).put_heap(...) | | §12.9.7 |
| table(...).heap_ptr() | | §12.9.5 |
| table(...).heap_length() | ヒープ領域のバイト長 | §12.8.5 |
| table(...).resize_heap() | ヒープ領域サイズの変更 | §12.9.8 |
| table(...).reverse_heap_endian() | ヒープ領域のエンディアン変換 | §12.9.9 |
| table(...).reserved_area_length() | 予約領域のバイト長 | §12.9.10 |
| table(...).resize_reserved_area() | 予約領域サイズの変更 | §12.9.11 |
| [**fits_table_col class**] | | |
| table(...).col(...).heap_is_used() | 可変長配列かどうかを判定 | §12.8.9 |
| table(...).col(...).heap_type() | 可変長配列におけるデータ型 | §12.8.10 |
| table(...).col(...).heap_bytes() | 可変長配列の 1 要素のバイト長 | §12.8.16 |
| table(...).col(...).max_array_length() | 可変長配列の長さの最大 | §12.8.17 |
| table(...).col(...).array_length(long) | 引数で指定された行での可変長配列の長さ | §12.8.18 |
| table(...).col(...).array_heap_offset(...) | 当該可変長配列のヒープ上での位置を返す | §12.9.21 |
| table(...).col(...).assign_arrdesc(...) | 可変長配列の配列記述子をセット | §12.9.32 |

# 5 チュートリアル

簡単な使用例を示しながら，お気楽極楽な SFITSIO の世界をご案内します.

## 5.1 最初のおまじない

SFITSIO を使う場合，ソースファイルの先頭に必ず次のように書いてください.

```
#include <sli/fitscc.h>
using namespace sli;
```

## 5.2 ファイルの読み書き

ファイルの読み書きは，それぞれ read_stream() メンバ関数 (§12.3.1)，write_stream() メンバ関数 (§12.3.3)，で行ないます.

次の例では，コマンドラインの最初の引数で指定されたファイルを読み込み，HDU の情報を出力し，コマンドラインの第二引数で指定されたファイルに読み込んだ内容をすべて書き出します．わかりやすくするために，まずはエラー処理を省略したコードを示します.

```
#include <stdio.h>
#include <sli/fitscc.h>
using namespace sli;

int main( int argc, char *argv[] )
{
    long i;
    fitscc fits;                                /* FITS オブジェクト */
    fits.read_stream(argv[1]);                  /* ファイル読み込み */
    for ( i=0 ; i < fits.length() ; i++ ) {     /* HDU 情報を表示 */
        printf("HDU %ld : hduname = %s\n", i, fits.hduname(i));
    }
    fits.write_stream(argv[2]);                 /* ファイル書き込み */
    return 0;
}
```

コンパイルし，実行します．s++を使うとコンパイルが楽に行なえます.

```
$ s++ fits_io.cc -lsfitsio
g++ -I/usr/local/include -L/usr/local/lib -Wall -O fits_io.cc -o fits_io -lsfits
io -lsllib -lz -lbz2 -lreadline -lcurses
$ ./fits_io in.fits out.fits
HDU 0 : hduname = Primary
```

次は，しっかりエラー処理を行なったコードです．SFITSIO では，基本的にプログラマがメモリを確保する必要がないため，エラー処理は最小限で済むようになっていますが[26]，ファイル入出力については，メンバ関数の返り値でエラー処理を行なう必要があります.

[26] SFITSIO では，メモリが確保できなかった場合，原因を標準エラー出力に表示して abort するようになっています.

```
#include <stdio.h>
#include <sli/fitscc.h>
using namespace sli;

int main( int argc, char *argv[] )
{
    int return_status = -1;
    fitscc fits;                            /* FITS オブジェクト */
    ssize_t sz;

    if ( 1 < argc ) {
        long i;
        const char *in_file = argv[1];
        sz = fits.read_stream(in_file);     /* ファイル読み込み */
        if ( sz < 0 ) {                     /* エラー処理 */
            fprintf(stderr,"[ERROR] fits.read_stream() failed\n");
            goto quit;
        }
        /* HDU 情報を表示 */
        for ( i=0 ; i < fits.length() ; i++ ) {
            fprintf(stderr,"HDU %ld : hduname = %s\n",i,fits.hduname(i));
        }
    }
    if ( 2 < argc ) {
        const char *out_file = argv[2];
        sz = fits.write_stream(out_file);   /* ファイル書き込み */
        if ( sz < 0 ) {                     /* エラー処理 */
            fprintf(stderr,"[ERROR] fits.write_stream() failed\n");
            goto quit;
        }
    }
    return_status = 0;
quit:
    return return_status;
}
```

ソースパッケージのサンプルコード sample/read_and_write.cc もご覧ください.

## 5.3 ネットワーク経由でリモートの FITS ファイルへ直接アクセス

ネットワーク経由でファイルへアクセスする場合も，read_stream() や write_stream() がそのまま使えます．引数に，http:// あるいは ftp:// で始まるパス名を指定します.

次の例は，Web サーバから読み込む例です.

```
    sz = fits.read_stream("http://www.xxx.jp/fits_data/foo.fits.bz2");
```

Web サーバに対しては読み込みしかできませんが，FTP サーバに対しては，書き込みも行なう事ができます.

```
    sz = fits.write_stream("ftp://user:passwd@myhost.jp/home/user/foo.fits.gz");
```

このように，FTP の場合はユーザ名とパスワードを含める事ができます．ユーザ名とパスワードを省略した場合は，匿名でアクセスします.

## 5.4 コマンドとパイプ接続して FITS を読み書き (圧縮・ネットワークツールとの併用)

各種コマンドが標準入力，標準出力での入出力をサポートしていれば，どんな特殊なネットワークや圧縮ファイルでもパイプ経由で FITS を読み書きできます.

access_stream()，accessf_stream() (§12.3.4) を使うと，コマンドラインツール経由でファイルにアクセスする事ができます．この場合は，Perl の open() に似た方法でファイル名を含むコマンドを引数に指定します.

次の例は，HEASARC の特殊な圧縮ファイルを funpack コマンドで読むものです.

```
sz = fits.access_stream("funpack -S foo0.fits.fz |");
```

次の例は fpack コマンドを使って特殊な圧縮ファイルを作成します.

```
sz = fits.access_stream("| fpack - foo1.fits.fz");
```

HTTP over SSL を使って FITS を読む例が §12.3.4 の EXAMPLE-2 にあります.

## 5.5 ヘッダへのアクセスの基本

ポイントは API の形が FITS ファイルの構造をそのまま反映しているという事です．例えば，Primary のヘッダの TELESCOP の値を読む場合,

```
printf("TELESCOP = %s\n", fits.hdu("Primary").header("TELESCOP").svalue());
```

と書きます．値の読み出しには dvalue()，lvalue()，llvalue()，bvalue()，svalue() のいずれかを使います (§12.4.4〜)．それぞれ，double，long，long long，bool，const char * の各型に対応します.

逆に，書き込む場合は，新規ヘッダレコード追加の場合も値やコメントの更新の場合も，assign() や assign_comment() を使います (§12.4.9〜).

```
fits.hdu("Primary").header("TELESCOP").assign("HST")
                                      .assign_comment("Telescope Name");
fits.hdu("Primary").headerf("OBJECT%d",n).assignf("%s-%d",obj[i],j)
                    .assignf_comment("Name of the object No.%d",n);
```

のように書きます．libc の printf() 関数の書き方がいたるところで使えるので，大変便利です．HISTORY などの記述レコードの追加も簡単で，header_append() メンバ関数 (§12.4.28) に 2 つの引数を与え,

```
fits.hdu("Primary").header_append("HISTORY","step-0: done.");
```

とします．なお，Primary HDU の場合は必ずイメージを扱う HDU ですので，fits.hdu( ... ) は fits.image( ... ) としてもかまいません.

ソースパッケージのサンプルコード sample/read_header.cc もご覧ください.

## 5.6 ヘッダのキーワード検索 (POSIX 拡張正規表現)

WCS を扱う FITS ヘッダを操作する時などに威力を発揮するのが，正規表現によるキーワード検索です．次のコードは，プライマリ HDU のヘッダのキーワード名が CRVAL1 あるいは CRVAL2 から始まるレコードを検索する例です．header( ... ).keyword() と header( ... ).value() (§12.4.17) を使って，キーワードと生のヘッダ値を表示します.

```
fits_image &primary = fits.image("Primary");
long i = 0;
while ( 0 <= (i=primary.header_regmatch(i,"^CRVAL[1-2]")) ) {
    printf("%s = %s\n",primary.header(i).keyword(),
                       primary.header(i).value());
    i++;
}
```

この例で,「fits_image &primary = …」は参照またはエイリアス呼ばれる C++で導入された変数で，マクロのような「別名」を定義する極めて単純な仕組みです．参照を使うと，コードをより短かく書く事ができます．詳しくは，§6.3をご覧ください．

## 5.7 ヘッダの編集

ヘッダの初期化 (§12.4.25)，ヘッダレコードの追加 (§12.4.27)，挿入 (§12.4.29)，削除 (§12.4.31) を行なうメンバ関数が用意されています．ヘッダ編集のためのメンバ関数では，構造体を用いて多くのヘッダレコードを一度に扱う事ができます (§5.9の例を参照してください)．

次のコードは，一方の FITS のヘッダレコードのすべての内容を，もう 1 方にコピーする例です．

```
fits_out.image("Primary")
        .header_append_records( fits_in.image("Primary").header() );
```

このように，引数を省略した形 .header() は，ヘッダ全体を表現しており，同様にして，header_init() メンバ関数 (§12.4.25)，header_insert_records() メンバ関数 (§12.4.29) に与える事ができます．

ヘッダレコードのすべての内容ではなく，1 つのヘッダレコードをコピーする場合は，次のようにします．

```
fits_out.image("Primary")
        .header_append( fits_in.image("Primary").header("TELESCOP") );
```

## 5.8 イメージデータへのアクセス

画素の大きさは，col_length()，row_length()，layer_length() で調べる事ができます (§12.6.7〜)．

```
printf("カラムの数 : %ld\n",fits.image("Primary").col_length());
printf("ロウの数   : %ld\n",fits.image("Primary").row_length());
printf("レイヤの数 : %ld\n",fits.image("Primary").layer_length());
```

読み出しには dvalue()，lvalue()，llvalue() のいずれかを使い，書き込みには assign() を使います (§12.6.10〜)．これらは，ヘッダの BZERO，BSCALE の値による変換もやってくれます．

dvalue() を使うと，イメージデータの型にかかわらず，double で読む事ができます．座標値は 0 から始まります．

```
double pixel_val;
pixel_val = fits.image("Primary").dvalue(x,y,z);   /* 読み */
fits.image("Primary").assign(pixel_val,x1,y1,z1);  /* 書き */
```

もちろん，整数値で読み書きする事もできます．lvalue()，llvalue() でそれぞれ long，long long を返してくれます．

```
    long pixel_val;
    pixel_val = fits.image("Primary").lvalue(x,y,z);   /* 読み */
    fits.image("Primary").assign(pixel_val,x1,y1,z1);  /* 書き */
```

## 5.9 新規イメージ FITS の作成

CASSIOPEIA という望遠鏡でとったデータのイメージファイルを作る例を紹介します．コンパイルできるサンプルコードを配布パッケージの sample ディレクトリの create_image.cc, create_image_and_header.cc に用意していますので，そちらもご利用ください．

イメージ HDU の作成は，append_image() メンバ関数 (§12.3.16) を使います．次の例では，1024 × 1024 の double 型で作ってみます．

```
    fitscc fits;
    fits::header_def defs[] = { {"TELESCOP", "'CASSIOPEIA'", "Telescope name"},
                                {"OBSERVAT", "'NAOJ'", "Observatory name"},
                                {"RA",       "", "[deg] Target position"},
                                {"DEC",      "", "[deg] Target position"},
                                {"COMMENT",  "------------------------------"},
                                {NULL} };
    /* Image HDU (Primary) の作成 */
    fits.append_image("Primary",0, FITS::DOUBLE_T,1024,1024);
    /* ヘッダの初期化 */
    fits.image("Primary").header_init(defs);
```

これだけで準備は完了．SFITSIO では，この例のように構造体でヘッダをあらかじめ用意しておく事ができ，header_init() メンバ関数 (§12.4.25) などで一度に多くのヘッダレコードを登録する事ができます．

image().assign() を使ってピクセル値をセットできます．必要な場合は，image().assign_bzero(), image().assign_bscale() で BZERO，BSCALE をセットし (§12.6.14〜)，image().fill() でゼロに初期化します (§12.6.28)．

```
    fits.image("Primary").assign_bzero(32768);
    fits.image("Primary").assign_bscale(1);
    fits.image("Primary").fill(0);
```

## 5.10 イメージデータのコピー&ペースト

矩形の領域のコピーとペーストを行なうためのメンバ関数，copy() と paste() が用意されています (§12.6.34〜)．次に示すのは，座標 (0,0) から (99,99) までの矩形領域を，座標 (100,100) へコピーする例です．

```
    fits_image copy_buf;
    fits.image("Primary").copy(&copy_buf, 0, 100, 0, 100);
    fits.image("Primary").paste(copy_buf, 100, 100);
```

プログラマが作った fits_image のオブジェクト copy_buf がコピーバッファとなります．その copy_buf オブジェクトをメンバ関数に与えて，コピー&ペーストを行なうわけです．コピーバッファはいくつでも持つ事ができますし，オブジェクト間のコピー&ペーストも簡単です．

paste のかわりに add，subtract，multiply，divide を使うと，足し算，引き算，掛け算，割り算もできます (§12.6.29〜).

上記の例のコピーバッファとして使っている copy_buf オブジェクトは，§12.6のすべてのメンバ関数を使う事ができます．例えば，

```
v = copy_buf.dvalue(x0,y0);
copy_buf.assign(v,x1,y1);
```

のように，値の読み書きもできますし，各種編集も同様に行えます．つまり，copy_buf オブジェクトは，fits オブジェクト (FITSCC クラスのオブジェクト) の管理下にないイメージ HDU というわけです.

さて，copy_buf オブジェクトを FITS ファイルに保存する事もできます．その場合には，FITSCC クラスの新しいオブジェクトを作って，それに copy_buf オブジェクトの内容を登録してファイルに保存します.

```
fitscc new_fits;
new_fits.image("Primary").swap(copy_buf);
new_fits.write_stream("copy_buffer.fits");
```

この例では，新しいオブジェクト new_fits にイメージデータを含まないプライマリ HDU を作り (new_fits.image("Primary") で，プライマリ HDU が存在しない場合は作成します)，copy_buf と内容を交換しています．swap() メンバ関数を使うかわりに，new_fits.append_image(copy_buf); とする方法もありますが，swap() メンバ関数を使う場合に比べてメモリを 2 倍消費するので，イメージデータが大きい場合にはお勧めできません.

## 5.11 イメージデータの型変換と高速アクセス

オブジェクト内部データへのポインタ変数を使った高速アクセスも可能です．ただし，image().resize() メンバ関数 (§12.6.22) を使った場合などで画素のサイズが変更になった場合は，データ配列のアドレスが変わるので注意が必要です.

高速アクセスのためには，値を読む時の BZERO 値と BSCALE 値による変換がいらないように内部データを変換してしまうと良いでしょう．変換には，image().convert_type() メンバ関数 (§12.6.13) を使います．例えば，次のようにします.

```
fits.image("Primary").convert_type(FITS::DOUBLE_T);
```

以上で，どの形式のデータも BZERO が 0，BSCALE が 1 の double 型に変換されます．この後，image().double_t_ptr() メンバ関数等 (§12.7.2) を使って，データのアドレスを得ます (それぞれの型に対応したメンバ関数があります).

```
fits::double_t *ptr:
ptr = fits.image("Primary").double_t_ptr();
```

ここまでくれば，「ptr[x + y * col_length]」の形でアクセスするだけです.

## 5.12 WCSTools の libwcs との連携

WCSTools[27] は WCS(World Coordinate System) をはじめとした，天文学のデータを効率良く扱うために開発された C 言語ライブラリです．FITS ファイルや天体カタログファイルにも対応してお

[27] http://tdc-www.harvard.edu/wcstools/

り，SAO の Douglas J. Mink 氏が開発・メンテナンスしています.

ここでは，SFITSIO を libwcs と組み合わせると，簡単に WCS を扱える事を示してみます.

まず，次のようにヘッダファイルをインクルードします.

```
#include <sli/fitscc.h>
#include <math.h>
#include <libwcs/wcs.h>
using namespace sli;
```

次に，main() 関数の例を示します．始めに，foo.fits.gz という FITS ファイルを読み込み，wcsinitn() 関数を使って，プライマリ HDU について，FITS ヘッダから WorldCoor 構造体のオブジェクトを wcs に作成しています．この時に，プライマリ HDU のすべてのヘッダレコードを 1 つの文字列として返してくれる header_formatted_string() メンバ関数 (§12.4.35) を利用しています．その後，pix2wcs() 関数を使ってピクセル座標 (0,0) について世界座標 (lon, lat) を求め，さらに wcs2pix() 関数で逆の変換をしています．最後に当該ピクセルの値を表示し，wcs が使っているメモリを開放しています.

```
int main( int argc, char *argv[] )
{
    fitscc fits;                                    /* FITS object */
    struct WorldCoor *wcs;                          /* WCS structure */
    double lon,lat, x,y, v;
    int off;

    /* read all data from fits file */
    fits.read_stream("foo.fits.gz");

    /* create alias 'pri' to Primary HDU */
    fits_image &pri = fits.image("Primary");

    /* initialize wcs structure */
    wcs = wcsinitn(pri.header_formatted_string(), NULL);

    /* convert pix -> wcs */
    x = 1.0;  y = 1.0;
    pix2wcs(wcs, x, y, &lon, &lat);
    printf("ra=%.8f dec=%.8f\n",lon,lat);

    /* convert wcs -> pix */
    wcs2pix(wcs, lon, lat, &x, &y, &off);
    printf("x=%.8f y=%.8f\n",x,y);

    /* read value of pixel (1-indexed) */
    v = pri.dvalue((long)floor(x-0.5),(long)floor(y-0.5));
    printf("value=%.15g\n",v);

    /* free wcs structure */
    wcsfree(wcs);

    return 0;
}
```

実際のコードでは wcsinitn() を使った後，iswcs() 関数で WorldCoor 構造体のオブジェクトが正しくセットされたかどうかをチェックします (iswcs() 関数は，正しくセットされた場合は 1，そうでない場合は 0 を返します).

WCSToolsのこれ以上の情報については，ヘッダファイル`wcs.h`等を参照してください．WCSToolsのヘッダファイルには，かなりしっかりとコメントが記載されており，それらを読むだけでもかなりの事ができるようになるはずです．

ソースパッケージのサンプルコード`sample_wcs/wcs_test.cc`もご覧ください．

## 5.13 WCSLIBとの連携

WCSLIB[28]は、WCS(World Coordinate System)を扱うためにMark Calabretta博士により開発されたライブラリです．SFITSIOをWCSLIBと組み合わせる事により，簡単にWCSを扱うことができます．ここでは，WCSLIBとSFITSIOを組み合わせた簡単なプログラムの例を紹介します．このプログラムではWCSヘッダを持った既存のFITSファイル(オブジェクト`in_fits`に読み込む)に対して，そのあるピクセルに対応する天球上の座標値を求め，新たなFITSファイル(オブジェクト`out_fits`に作成)にWCSヘッダをつけてピクセル値を求めてみます．

まず，wcslibを使うには，次のようにヘッダファイルをインクルードします．

```
#include <sli/fitscc.h>
#include <sli/tstring.h>
#include <wcshdr.h>
#include <wcs.h>
```

main()関数などに以下のように書いて，ファイル出力のためのオブジェクトout_fitsを作ります．

```
fitscc out_fits;
struct *wcs_out;
tstring headerall;
int status=0, nrecords, relax=1, nreject, nwcs, ctrl=0, anynul;
out_fits.append_image("Primary",0,
                      FITS::FLOAT_T,1024,1024); /* 1024x1024 の FLOAT イメージです */
fits_image &outfitspri = outfits.image("Primary");
outfitspri.header("RADESYS").assign("FK5").assign_comment("Coordinate System");
outfitspri.header("EQUINOX").assign(2000.0).assign_comment("Equinox");
outfitspri.header("CTYPE1").assign("RA---TAN"); /* Tangential projection */
outfitspri.header("CTYPE2").assign("DEC--TAN"); /* Tangential projection */
outfitspri.header("CRPIX1").assign(512.5);      /* 基準点となるピクセルの座標値 */
outfitspri.header("CRPIX2").assign(512.5);      /* 基準点となるピクセルの座標値 */
outfitspri.header("CRVAL1").assign(0.0);        /* 基準点の赤経 */
outfitspri.header("CRVAL2").assign(0.0);        /* 基準点の赤緯 */
outfitspri.header("CDELT1").assign(-0.01);      /* 座標値の増分（赤経は左向に増加）*/
outfitspri.header("CDELT2").assign(0.01);       /* 座標値の増分（赤緯は上向に増加）*/
headerall = outfitspri.header_formatted_string();
nrecords = headerall.length()/80;
wcspih((char*)headerall.cstr(), nrecords, relax, ctrl, &nreject, &nwcs, &wcs_out);
wcsprt(wcs_out);
```

ここまでで，オブジェクトout_fitsを生成し，WCSヘッダを定義してその値を構造体`wcs_out`に入れたことになります．WCSヘッダを構造体に入れるための関数がwcspih()関数で，その最初の引数にはWCS関連のヘッダすべてを含んだ1つの文字列を与える必要があります．この文字列には，FITSファイルに書かれたバイトイメージを与える必要がありますが，SFITSIOではこの例のようにheader_formatted_string()メンバ関数(§12.4.35)を使うとヘッダのバイトイメージを得る事ができます(tstringクラスについては，APPENDIX3(§15)で解説しています)．最後にwcsprt()関数で，その構造体の中身を画面に出力しています．

[28] http://www.atnf.csiro.au/people/mcalabre/WCS/WCSLIB

同様に，オブジェクト in_fits を作ってファイルを入力し，WCS ヘッダを構造体 wcs_in に入れます．

```
fitscc in_fits;
struct wcsprm *wcs_in;
in_fits.read_stream("user_image_file.fits");
headerall = in_fits.image("Primary").header_formatted_string();
nrecords = headerall.length()/80;
wcspih((char*)headerall.cstr(), nrecords, relax, ctrl, &nreject, &nwcs, &wcs_in);
wcsprt(wcs_in);
```

オブジェクト in_fits のあるピクセル座標 (x_out, y_out) に対応する天球上の座標値 (world[0][0], world[0][1]) を求め，それがオブジェクト out_fits では，どのピクセル座標に対応するかを計算し，(pixcrd_in[0][0], pixcrd_in[0][1]) に代入します．

```
double pixcrd_in[1][2], pixcrd_out[1][2], imgcrd[1][2];
double phi[1], theta[1], world[1][2];
double x_out, y_out, x_in, y_in;
pixcrd_out[0][0] = (double)x_out;
pixcrd_out[0][1] = (double)y_out;
wcsp2s(wcs_out, 1, 2, pixcrd_out[0], imgcrd[0], phi, theta, world[0], &status);
wcss2p(wcs_in, 1, 2, world[0], phi, theta, imgcrd[0], pixcrd_in[0], &status);
x_in = (float)pixcrd_in[0][0];
y_in = (float)pixcrd_in[0][1];
```

wcsp2s() 関数でピクセル座標から世界座標へ，wcss2p() 関数で世界座標からピクセル座標に変換している訳です．ここでは，wcspih(), wcsprt(), wcsp2s(), wcss2p() という関数を紹介しました．最低限，これらの関数だけを知っていれば，WCS に関連した計算ができます．また，他に便利な関数として，wcsset()，wcs_errmsg() があります．前者は，wcsset(wcs_out) のようにして，構造体をリセットするときに使います．後者は，wcs_errmsg(status) として，status に対応するエラーメッセージの文字列を取得します．

## 5.14 アスキーテーブル・バイナリテーブルへのアクセス

SFITSIO では，アスキーテーブルもバイナリテーブルも全く同じメンバ関数で扱う事ができます．

まず，テーブルの大きさを調べる方法です．table().row_length() メンバ関数 (§12.8.4) と table().col_length() メンバ関数 (§12.8.3) で得る事ができます．次に示すのは，「EVENT」という名のバイナリテーブルの大きさを表示する例です．

```
printf("カラムの数 : %ld\n",fits.table("EVENT").col_length());
printf("ロウの数   : %ld\n",fits.table("EVENT").row_length());
```

テーブルデータへのアクセスも，やはり FITS ヘッダの場合と同じような書き方をします．やり方は単純で，fits.table(HDU 名).col(カラム名).…のようにアクセスします．HDU やカラムに名前が無い場合は，0 から始まる数字を指定する事もできます．

値の読み出しには dvalue()，lvalue()，llvalue()，bvalue()，svalue() のいずれかを使い，書き込みには assign() を使います (§12.8.20〜)．これらは，ヘッダの TZERO$n$，TSCAL$n$ の値による変換もやってくれます．

dvalue() を使うと，カラムのデータの型にかかわらず，実数値 (double) で読む事ができます．行番号は 0 から始まります．次に示すのは，「EVENT」という名のバイナリテーブルの「TIME」というカラムの値を読み書きする例です．

```
double val;
val = fits.table("EVENT").col("TIME").dvalue(row_index0);  /* 読み */
fits.table("EVENT").col("TIME").assign(val,row_index1);    /* 書き */
```

このように，読み出しの場合は引数に行番号を指定し，書き込みの場合は引数に，値，行番号の順に指定します．

整数値や論理値で読み書きする事もできます．lvalue()，llvalue()，bvalue() でそれぞれ long，long long，bool の値を返してくれます．文字列で取り出すには，svalue() を使います．svalue() を使うと，値が数値の場合にはヘッダの TDISP$n$ の指定があればそれ従ってフォーマットした文字列を得る事ができます．次の例では，カラム「TIME」を TDISP$n$ の指定に従って表示しています．

```
const char *sval;
sval = fits.table("EVENT").col("TIME").svalue(row_index0);  /* 読み */
printf("%s\n",sval);                                        /* 標準出力へ */
```

カラム中に複数の要素が存在する場合，例えば TFORM$n$ = '8J' のような場合は，カラム中に 8 つの要素がある事を意味します．この要素の個数は，table( ... ).col( ... ).elem_length() で得る事ができます (§12.8.13)．どの要素を取り出すかは，dvalue()，lvalue()，llvalue()，bvalue()，svalue() いずれの場合も 2 つめの引数で指定できます．assign() で書き込む場合は，3 つめの引数で指定できます．次の例は，カラム「STATUS」の全要素についてすべての行を表示するものです．

```
long i, j, nrow, nel;
const char *sval;
nrow = fits.table("EVENT").row_length();                    /* 行の個数 */
nel = fits.table("EVENT").col("STATUS").elem_length();      /* 要素の個数 */
for ( i=0 ; i < nrow ; i++ ) {
    for ( j=0 ; j < nel ; j++ ) {
        sval = fits.table("EVENT").col("STATUS").svalue(i,j);
        printf("%s ",sval);
    }
    printf("\n");
}
```

## 5.15 バイナリテーブルの新規作成

ASTRO-X という天文衛星のデータのためのバイナリテーブルを作る例を紹介します．コンパイルできるサンプルコードを配布パッケージの sample ディレクトリの create_bintable.cc に用意していますので，そちらもご利用ください．

3 つのカラム (倍精度浮動小数点型，32-bit 整数型，文字列型) を持つテーブルを作ります．

```
fitscc fits;
const fits::table_def def[] = {
  /* ttype,comment,         talas,telem,tunit,comment,                 */
  /*                                          tdisp,  tform, tdim   */
  { "TIME","satellite time",   "",  "", "s","", "F16.3", "1D", "" },
  { "STATUS","status",         "",  "", "","",  "",      "8J", "" },
  { "NAME","",                 "",  "", "","",  "",  "128A16", "(4,2)" },
  { NULL }
};
/* バイナリテーブルの作成 (HDU 名は"EVENT") */
fits.append_table("EVENT",0, def);
```

まず，カラムの定義を構造体を使って用意し，append_table() メンバ関数 (§12.3.17) でバイナリテーブルを作ります．バイナリテーブルは FITS 規約では Primary HDU にはなれないので，この場合は，イメージデータを含まない Primary HDU が自動的に作られます．

その後，次のようにテーブルの行を確保します．

```
fits.table("EVENT").resize_rows(256);
```

## 5.16 アスキーテーブルの新規作成

アスキーテーブルの作り方はバイナリテーブルとほとんど同じですが，構造体の tdisp，tform の指定には注意が必要です．アスキーテーブルの場合は，構造体の tdisp に FITS ファイルの TFORM$n$ に書かれる文字列を指定し，構造体の tform に文字列幅を"$n$A"の形で指定します．

下記のコードは，3 つのカラムからなるテーブルを作ります．

```
fitscc fits;
const fits::table_def def[] = {
  /* ttype,comment,        talas,telem, tunit,comment, tdisp,  tform */
  { "PK","PK number",        "",   "",  "","",         "A9",   "9A" },
  { "RAH","Hours RA",        "",   "",  "h","",        "I2",   "3A" },
  { "RAM","Minutes RA",      "",   "",  "min","",      "F5.2", "6A" },
  { NULL }
};
/* アスキーテーブルの作成 (HDU 名は"PLN") */
fits.append_table("PLN",0, def, true);
```

まず，カラムの定義を構造体を使って用意し，append_table() メンバ関数 (§12.3.17) の最後の引数に true をつけてでアスキーテーブルを作ります．アスキーテーブルは FITS 規約では Primary HDU にはなれないので，この場合は，イメージデータを含まない Primary HDU が自動的に作られます．

その後，次のようにテーブルの行を確保します．

```
fits.table("PLN").resize_rows(256);
```

ソースパッケージのサンプルコード sample/create_asciitable.cc もご覧ください．

## 5.17 アスキーテーブル・バイナリテーブルの編集・インポート

カラムの追加・挿入・削除 (§12.8.46〜) や行の追加・挿入・削除 (§12.8.52〜) はもちろん，別のアスキーテーブル・バイナリテーブルからのインポート (§12.8.58) も簡単に行なえます．

次の例は，一方の FITS のテーブルのカラム DEC をそのままの内容で，もう 1 方にコピーする例です.

```
fits_out.table("EVENT").append_a_col( fits_in.table("SRC").col("DEC") );
```

今度は，2 つのテーブルを合体させる例を紹介します.

```
long orow_length = fits_out.table("EVENT").row_length();
fits_out.table("EVENT")
        .resize_rows( orow_length + fits_in.table("SRC").row_length() );
fits_out.table("EVENT")
        .import_rows( orow_length, true, fits_in.table("SRC") );
```

まず，resize_rows() メンバ関数で行の数を 2 つのテーブル分に広げています．その後，import_rows() メンバ関数 (§12.8.58) でテーブル SRC の内容すべてをテーブル EVENT 後半の余白に貼り付けます．import_rows() の引数は順に，貼り付けを開始する行番号，カラム名で一致したカラムを貼り付けるかどうか，源泉となるテーブルです.

カラム単位でインポートしたい場合には，table().col().import() メンバ関数 (§12.8.62) を使います.

## 5.18 HDU の編集

複数の FITS ファイルのイメージとアスキーテーブル・バイナリテーブルとを 1 つの FITS にまとめたり，処理後に不要な HDU を削除する，といった作業も SFITSIO なら一瞬です．.append_image() や .append_table() などのメンバ関数の引数に，fits.image("FOO") あるいは fits.table("BAR") を与える事で，内容を保ったまま Image HDU の追加 (§12.3.16)，Binary (Ascii) Table HDU の追加 (§12.3.17)，Image HDU の挿入 (§12.3.18)，Binary (Ascii) Table HDU の挿入 (§12.3.19) が簡単に行なえます.

次の例は，オブジェクト fits_in にあるテーブル EVENT をオブジェクト fits_out に内容そのままで追加します.

```
fits_out.append_table( fits_in.table("EVENT") );
```

FOO という名の HDU をオブジェクト fits_out から削除する例です.

```
fits_out.erase("FOO");
```

# 6 SFITSIOを使う前に知っておきたい事

SFITSIOはC++のライブラリですので，ユーザのみなさんもC++コンパイラを使う事になります．C++は基本的にはCの上位互換ですから，これまでのCの場合と同様にコードを書く事ができますので，ご安心ください．

ただ，全く難しくない事ですが，SFITSIOを使う前にちょっとだけ知っておきたい事があります．それは，C++の拡張にともなって微妙にCとの互換性が失われている部分と，難易度が低くて便利なC++の拡張機能などです．この章ではそれらを解説します．

## 6.1 NAMESPACE

C++で導入されたものに，namespaceがあります．これは，別々の人が同じ名前の関数や型を作ってしまうと困った事になる，という問題を回避するもので，要するにカテゴリ名のようなものです．SLLIBやSFITSIOでは，「sli」というnamespaceをつけており，例えば何かクラスを使う場合，正式には「`sli::fitscc my_fits;`」のように先頭に`sli::`をつけて使います．

しかし，SFITSIOを中心的に使っていく，という事であれば，いちいち`sli::`を書きたくないものです．その場合，

```
using namespace sli;
```

と書けば，それ以降では`sli::`を省略する事ができます．このマニュアルの使用例では，`sli::`を省略しています．C++が初めての方は，「`#include <...>`」したら「`using namespace sli;`」する，と覚えておけば良いでしょう．

## 6.2 NULLと0

多くの処理系では，Cの場合は

```
define NULL ((void*)0)
```

と定義されています．しかし，C++の場合，

```
define NULL (0)
```

と定義されます．

C++で`NULL`が`0`となっている理由は，C++の場合，ポインタ変数の型のチェックがCよりも厳格になっているからです．例えば，2つのポインタ変数`char *ptr0; void *ptr1;`があり，`ptr0`に`ptr1`を代入しようとするとエラーになります．しかし，`0`だけは「どこでもないアドレス」と定義しているので，`ptr0 = 0;`はエラーになりません．そういうわけで，C++では`NULL`は`0`になっているのです．

さて，C++では，クラスの持つメンバ関数は，同名でも引数が異なる場合があります．例えば，

```
int foo( int a );
int foo( char *p );
```

のような場合です．ここで，`hoge.foo(NULL)`あるいは`hoge.foo(0)`とすると，一体どちらの関数を使いたいのかコンパイラが理解できないという事態になります．この場合，`NULL`や`0`を使う場合にはキャストして，型を明示的に示す事が必要です．すなわち，

```
hoge.foo((char *)NULL);
hoge.foo((int)0);
```

のようにしなければなりません．

C++では,「NULL **や** 0 **を使う時はキャストする**」と覚えておけば間違いないでしょう．あるいは，NULL を捨ててしまって,「**いつも** 0 **をキャストして使う**」方法もあります.

## 6.3 参照

ヘッダやイメージへのアクセスのたびに，fits.hdu("Primary") や fits.image("Primary") を書くのは面倒です．もっと短かく書くには,「**参照**」を使います．参照は「**エイリアス**」とも呼ばれ，簡単に言えばある変数やオブジェクトの別名を作るという事です．マクロを変数やオブジェクトの別名として使う方法もありますが，参照の方がスマートな記述ができます.

参照は，C++で導入されたポインタ型に似た新しい型ですが，実は**ポインタ型よりも単純な事しかできません**．したがって，**使い方も簡単**です.

例えば，int a; という変数の参照を aref という名前で作る場合,

```
int &aref = a;
```

とします．参照は「エイリアス (別名)」とも呼ばれる事もありますが，まさしくそのように振舞います．例えば,

```
aref = 10;
```

とすると，a に 10 が代入されますし,

```
int b = aref;
```

とすると，b に a の値が代入されます．また,

```
int *p = &aref;
```

とすると，p に a のアドレスが代入されます.

このように，参照は変数の「別名」を提供するだけの簡単な仕組みであり，ポインタ変数のように * がたくさんついて考え込む事もありませんし，NULL が入る事もありません．NULL が入らないのは,

```
int &aref;
```

のような，御本尊が存在しない参照を作る事は禁止されているからです.

SFITSIO の場合，メンバ関数が返す参照を，ユーザが用意した参照変数にコピーする，といった使い方になる場合がほとんどだと思います．例えば，メンバ関数 fits.hdu() は「fits_hdu &」という参照を返すわけですが，この場合，受け手側で同じクラスの参照を作ってコピーすれば，fits.hdu("Primary") や fits.image("Primary") の代用になり，コードを短かく書く事ができます．例えば，次のようにします.

```
fits_hdu &primary = fits.hdu("Primary");
printf("TELESCOP = %s\n",primary.header("TELESCOP").svalue());
```

あるいは,

```
fits_image &primary = fits.image("Primary");
printf("TELESCOP = %s\n",primary.header("TELESCOP").svalue());
```

と書けます.

参照は宣言時に値を代入しない使い方はエラーになります．というのは,「エイリアス」は日本語では「別名」ですから，御本尊のない「別名」はエラーになるのです．ご注意ください.

### 6.4 try & catch

この部分については，本格的なコーディングを目指さないのであれば，読み飛ばしてくださってかまいません．

try{} と catch(){} は，C++で導入された「例外」を扱うための構文です．SFITSIO は,「**メモリ確保失敗**」などのユーザが与えた引数と無関係な**致命的な問題**が生じた場合には,「例外」を発生させます．この「例外」は，ユーザのコードでは，try{} と catch(){} で捉える事ができます．SFITSIO 場合は必ず err_rec 型のメッセージを伴った例外が発生するので，これを捉えたい場合は，

```
try {
    /* イメージのバッファサイズを変更 */
    fits.image("Primary").resize(0,very_big_size);
    return_status = 0;
}
catch ( err_rec msg ) {
    fprintf(stderr,"[EXCEPTION] function=[%s::%s] message=[%s]\n",
            msg.class_name, msg.func_name, msg.message);
    return_status = -1;
}
```

のようにします．なお，try{} と catch( err_rec msg ){} を使っていない場合に例外が発生すると，abort() 関数が呼ばれ，プログラムは終了します．

通常，このような致命的なエラーが発生した場合というのは，それ以上プログラムを続行できないケースがほとんどだと思います．したがって，例外発生時に abort() 関数が呼ばれる仕様で問題なければ，try{} と catch( err_rec msg ){} を使う必要はありません．

# 7 SFITSIO で的確に FITS を扱うためのヒント

## 7.1 SFITSIO のメモリ管理手法に対するコーディングとハードウェア選定指針

SFITSIO と SLLIB が提供するクラスは内部に参照カウンタを一切持たず[29]，必ず 1 オブジェクトにつき 1 つの内部バッファ(群) を持つという極めて単純な設計になっています．これは，`init()` メンバ関数や “=” 記号などにより新規オブジェクトに対して代入を行なうと，代入処理に必要なメモリ領域を即座に確保する事を意味します．

メモリのマッピングについても非常に単純で，`fits_image` クラスの画像用，および `fits_table_col` クラスのカラムデータ用ともに，**常に 1 次元のバッファ**となっています．

したがって，プログラマは C 言語におけるメモリ領域と同じ感覚で，オブジェクトが持つ内部バッファのアドレスを利用してコードを組む事ができます (もちろん，そのような使い方はパフォーマンスが必要な場合に限定すべきですが)．その一方で，バッファの途中に**データを挿入したり，途中データの一部分を削除して後部データを前へ詰めるような操作は，メモリ上とはいえ実行コストが小さくありません**．良好なパフォーマンスを得るためには，このような操作をできるだけ減らすようなメンバ関数の選択とコーディングが必要です．

次にハードウェアについてです．SFITSIO を使ったアプリケーションが扱う FITS ファイルのサイズに対して十分なメモリ容量が必要であるというのは，言うまでもない事と思います．アプリケーションが扱う最大の FITS ファイルと同程度のメモリ容量の空きを平均的に持たせておける程度を目安として，メモリをインストールする事をお薦めします．

SFITSIO ではメモリの再確保についても OS とプログラマに任せるという単純な方針により，`resize()` メンバ関数等でバッファの大きさ変更の必要が生じたら必ず `realloc()` を内部で呼び出します．常にバッファの大きさに余裕を持たせて `realloc()` の回数を減らすようにするのか，あるいは `realloc()` の回数は増えても省メモリに徹するのかは，プログラマ次第です．

さらに，SFITSIO や SLLIB には，動的に作られたオブジェクトのアドレスを返すメンバ関数は一切ありません．したがって，プログラマが動的にオブジェクトを作らない限り，delete やガーベージコレクタは必要ありません．

## 7.2 高速動作のためのテクニック

高速化のためのテクニックは，しばしば計算機のアーキテクチャに依存します．しかも，計算機のアーキテクチャは時代によって変化しますから,「高速化」は永遠に終わらない課題です[30]．また，あまりに高速化ばかりを追及すると，コードの可読性や安全性が失われる恐れがあります．したがって，チューニングを行なう場合には，今後の計算機アーキテクチャの見通しや他の評価軸とのバランスを考えて実施する必要があります．

とはいえ，高速動作のためには最終的なマシン語コードの総ステップ数を減らすという基本は変わらず，そのためにはプログラマがどのようにすべきなのかを，SFITSIO の内部実装をみながらここで少し触れておきたいと思います．

- **名前によるアクセスも高速に動作するように工夫しているが，相対的には実行コストは大きい**

  §1.1の課題 1 で示したように，SFITSIO では名前でヘッダやテーブルのカラムにアクセスでき

[29] ついでに，`static` なメンバ関数も一切ありません．

[30] 極端な例では，こんな事がありました．CPU にキャッシュが無い時代は，ループ中の条件分岐等を嫌って，条件によって自身の機械語コードを書き換えるというテクニックが流行りました．しかし，CPU にキャッシュが搭載されるようになると，それらのコードは動作不能になりました．

ます．SFITSIO 内部では，ヘッダもテーブルも単純なオブジェクト配列なので，パフォーマンスを落とさないようにするには名前から番号への高速変換が必要です．

そこで，**SFITSIO 側の工夫**として，キー文字列と番号との関係をツリー構造に展開する事で高速検索を実現している SLLIB の ctindex クラスを使っています (詳細は SLLIB のマニュアルの上級編をご覧ください)．

しかし，このようなアルゴリズムでがんばったとしても，直接番号で指定された場合に比べて実行コストは大きくなります．したがって，**プログラマ側での工夫**は不要なわけではなく，より高いパフォーマンスを得るには，名前から番号の変換は 1 回だけにして，後は直接番号で指定したり，参照を使ったコードを書く必要があります．

- **インラインやマクロによるコード展開を活用しているメンバ関数を使う**

  画像処理のためのメンバ関数 (§4.7.16) では，`for()` ループの中にできるだけ条件分岐が入らないように，マクロでコードを展開して高速化を図っています．

  現在の SFITSIO では，ピクセル等のサイズを返すものと超低レベル API についてはインラインメンバ関数としています．インラインメンバ関数は，SFITSIO のバージョンアップにあわせて，徐々に増やしていく予定ですので，バージョンアップごとにヘッダファイルの内容をチェックしましょう．

- **メンバ関数の呼び出し回数を減らす**

  多重ループの内側などでは，メンバ関数呼び出しのオーバヘッドが問題になる事があります．SFITSIO では，インラインを使うなどして気をつかって実装していますが，まだまだ改良の余地が残されているので，今後，さらなる高速化を図っていく予定です．

  **プログラマ側**でもメンバ関数の呼び出しを少なくする事は，より良いパフォーマンスを得るには有効です．次のようにうまく参照を使いましょう．

```
const fits_table &tbl = fits.table("FOO");
const fits_table_col &col = tbl.col("BAR");
for ( j=0 ; j < tbl.row_length() ; j++ ) {
    printf("[%s]", col.svalue(j));
}
```

# 8 テンプレート機能

SFITSIO には，曖昧な文法で FITS の内容を定義できるテンプレートファイルから，データが入っていない新規 FITS ファイル (fitscc のオブジェクト) を作成する「テンプレート機能」を持っています．このテンプレートファイルは，FITS ヘッダに類似したテキスト形式のファイルで，普通のテキストエディタで作成します．CFITSIO にもほぼ同じテンプレート機能があり，様々な FITS フォーマットを管理する等の目的で使われています．

## 8.1 Image HDU の作成例

まず，プライマリ HDU のみを作成するテンプレートファイルの例を次に示します．

```
#
# Test for Primary only
#
NAXIS2 = 16
NAXIS1 = 16
BITPIX = 32
FMTTYPE = ASTRO-X xxx image format
CHECKSUM =
DATASUM  =
COMMENT ----------------------------------------------------------------
EQUINOX = 2000.0
CTYPE1  = RA---TAN
CTYPE2  = DEC--TAN
CRPIX1  = 0.0
CRPIX2  = 0.0
CRVAL1  = 0.0
CRVAL2  = 0.0
CDELT1  = 0.1
CDELT2  = 0.1
PC1_1   = 1.0
PC1_2   = 0.0
PC2_1   = 0.0
PC2_2   = 1.0
COMMENT ----------------------------------------------------------------
MESSAGE = 'FITS (Flexible Image Transport System) format is &' / In SFITSIO,
CONTINUE  'defined in "Astronomy and Astrophysics", volume 376, &' / this
CONTINUE  'page 359; bibcode: 2001A&A...376..359H' / message is not written &
CONTINUE  automatically.
```

テンプレートでは，キーワードの位置,「=」記号[31] の位置，値の位置，コメントの位置は順序があっていれば任意で，横方向の長さ制限はなく，長い文字列値では「CONTINUE」はあってもなくても良いというかなり緩いルールで記述できます．なお，先頭が「#」で始まる行はコメント文であり，この部分は無視されます．

では，このテンプレートから作られた FITS ファイルのヘッダを見てみましょう．

[31] CFITSIO のテンプレート機能では「=」が省略できますが，SFITSIO では省略できません．

```
SIMPLE  =                    T / conformity to FITS standard
BITPIX  =                   32 / number of bits per data pixel
NAXIS   =                    2 / number of data axes
NAXIS1  =                   16 / length of data axis 1
NAXIS2  =                   16 / length of data axis 2
EXTEND  =                    F / possibility of presence of extensions
FMTTYPE = 'ASTRO-X xxx image format' / type of format in FITS file
FTYPEVER=                    0 / version of FMTTYPE definition
CHECKSUM= 'ZlBRfj9OZjAOdj9O'   / HDU checksum : 2012-01-17T06:07:12
DATASUM = '0       '           / data unit checksum : 2012-01-17T06:07:12
COMMENT ----------------------------------------------------------------
EQUINOX =               2000.0 / equinox of celestial coordinate system
CTYPE1  = 'RA---TAN'           / type of celestial system and projection system
CTYPE2  = 'DEC--TAN'           / type of celestial system and projection system
CRPIX1  =                  0.0 / pixel coordinate at reference point
CRPIX2  =                  0.0 / pixel coordinate at reference point
CRVAL1  =                  0.0 / world coordinate at reference point
CRVAL2  =                  0.0 / world coordinate at reference point
CDELT1  =                  0.1 / world coordinate increment at reference point
CDELT2  =                  0.1 / world coordinate increment at reference point
PC1_1   =                  1.0 / matrix of rotation (1,1)
PC1_2   =                  0.0 / matrix of rotation (1,2)
PC2_1   =                  0.0 / matrix of rotation (2,1)
PC2_2   =                  1.0 / matrix of rotation (2,2)
COMMENT ----------------------------------------------------------------
MESSAGE = 'FITS (Flexible Image Transport System) format is defined in &'
CONTINUE  '"Astronomy and Astrophysics", volume 376, page 359; bibcode: &'
CONTINUE  '2001A&A...376..359H&'
CONTINUE  '' / In SFITSIO, this message is not written automatically.
END
```

このように，SFITSIO のテンプレート機能を使うと，FITS ヘッダを定義する時に，ユーザが書かなければならない項目を最低限で済ませる事ができます．上記のテンプレートの例では，SIMPLE, EXTEND，NAXIS キーワードおよび FITS の標準的なキーワードに対するコメントを省略していますが，SFITSIO は自動的に必要なキーワードとコメントとを可能な限り補完し，キーワードの順序を FITS 規約に合うように調整します．

## 8.2 Binary Table HDU の作成例

次はバイナリテーブルを作成する場合のテンプレートの例です．

```
#
# Primary
#
FMTTYPE  = 'ASTRO-X XXXX table format'
FTYPEVER = 101
EXTNAME  = 'Primary'
ORIGIN   = 'JAXA'
#
# 2nd HDU : Binary Table
#
XTENSION = BINTABLE
NAXIS2 = 24
PCOUNT = 65536
THEAP  = 32768
EXTNAME = XXXX_TEST
TXFLDKWD = 'TNOTE,TDESC,TLMIN,TLMAX,TDMIN,TDMAX'
 TTYPE#  = TIME
 TALAS#  = DATE
 TFORM#  = 1D
 TDESC#  = 'This is time' / description of this field
 TTYPE#  = COUNTER
 TFORM#  = 8J
 TLMIN#  = 1
 TLMAX#  = 16777216
 TDMIN#  = 0
 TDMAX#  = 0
 TTYPE#  = XNAME
 TFORM#  = 16A
 TTYPE#  = VLA
 TFORM#  = 1PJ(0)
 TNOTE#  = 'You can define variable length array in SFITSIO template&'
 CONTINUE  '' / annotation of this field
```

バイナリテーブルの場合は，プライマリ HDU に関する記述を完全に省略する事もできますし，この例のように定義したいキーワードのみ書く事もできます．

2 つめの HDU の定義は，必ず「XTENSION」キーワードから開始し，BITPIX，NAXIS，NAXIS1，NAXIS2，PCOUNT，GCOUNT，TFIELDS を省略可能です．なお，この例で使っている TXFLDKWD は，FITS 規約には存在しないテーブルカラムのキーワード (例えば，TLMIN$n$，TLMAX$n$ 等) を宣言するためのものです (§10.7を参照)．

カラム (フィールド) の定義については,「TTYPE#」のように，キーワードの数字のかわりに「#」記号を書くと，自動ナンバリングを行ないます．ただし，

```
 TTYPE#  = TIME
 TTYPE#  = COUNTER
 TFORM#  = 1D
 TFORM#  = 8J
```

のような並びの場合は，期待通りの動作をしないのでご注意ください．

では，テンプレートから作成した FITS ファイルのヘッダを見てみましょう．

```
SIMPLE  =                    T / conformity to FITS standard
BITPIX  =                   16 / number of bits per data pixel
NAXIS   =                    0 / number of data axes
EXTEND  =                    T / possibility of presence of extensions
FMTTYPE = 'ASTRO-X XXXX table format' / type of format in FITS file
FTYPEVER=                  101 / version of FMTTYPE definition
EXTNAME = 'Primary '           / name of this HDU
ORIGIN  = 'JAXA    '           / organization responsible for the data
END
```

```
XTENSION= 'BINTABLE'           / type of extension
BITPIX  =                    8 / number of bits per data element
NAXIS   =                    2 / number of data axes
NAXIS1  =                   64 / width of table in bytes
NAXIS2  =                   24 / number of rows in table
PCOUNT  =                65536 / length of reserved area and heap
GCOUNT  =                    1 / number of groups
TFIELDS =                    4 / number of fields in each row
THEAP   =                32768 / byte offset to heap area
EXTNAME = 'XXXX_TEST'          / name of this HDU
TXFLDKWD= 'TALAS,TDESC,TNOTE,TLMIN,TLMAX,TDMIN,TDMAX' / extended field keywords
TTYPE1  = 'TIME    '           / field name
TALAS1  = 'DATE    '           / aliases of field name
TFORM1  = '1D      '           / data format : 8-byte REAL
TDESC1  = 'This is time'       / description of this field
TTYPE2  = 'COUNTER '           / field name
TFORM2  = '8J      '           / data format : 4-byte INTEGER
TLMIN2  =                    1 / minimum value legally allowed
TLMAX2  =             16777216 / maximum value legally allowed
TDMIN2  =                    0 / minimum data value
TDMAX2  =                    0 / maximum data value
TTYPE3  = 'XNAME   '           / field name
TFORM3  = '16A     '           / data format : STRING
TTYPE4  = 'VLA     '           / field name
TFORM4  = '1PJ(0)  '           / data format : variable length of 4-byte INTEGER
TNOTE4  = 'You can define variable length array in SFITSIO template&'
CONTINUE  '' / annotation of this field
END
```

## 8.3 SFITSIO テンプレートのルール

SFITSIO のテンプレート機能は，次のルールにより動作します.

(1) テンプレートファイルは plain text とし，改行コードは UNIX または DOS 形式に限ります.

(2) 先頭が「#」記号の行は無視されます.

(3) タブ文字は空白に置き換えられます.

(4) キーワードの位置,「=」記号の位置，値の位置,「/」の位置，コメント文の位置は，それらの順序が FITS ヘッダと同様であれば任意です．ただし，空白またはタブ文字が行の先頭から 8 文字続いた場合は,「空白 8 文字キーワード」のレコードを作ります.

(5) テンプレートのカラム長 (横方向の長さ) に制限はありません.

(6) 長い文字列値は，1 行にそのまま記述しても，CONTINUE を使って複数行に分けて記述してもかまいません．ただし，CONTINUE を使った場合はテンプレート上の接続文字「&」の位置が FITS ファイルのヘッダに反映されるとは限りません.

(7) 文字列値は，数字や論理値 (T または F) と解釈できない場合はクォーテーションを省略できます．ただし，CONTINUE を伴なう場合は，クォーテーションを省略しない事を推奨します.

(8) 1 行に非常に長い COMMENT レコードが書かれた場合は，複数の COMMENT レコードに分けてヘッダに保存されます (できるだけ空白文字で分けるようにしています).

(9) 2 つめ以降の HDU に関する記述は，必ず「XTENSION」で始めなければなりません.

(10) 「XTENSION」キーワードとテーブルのカラム (フィールド) に関するキーワードを除き，キーワードの順序は任意です.

(11) テンプレート上のキーワードの順序は，FITS 規約に合わないものを除き，FITS ファイルでのヘッダに反映されます.

(12) Image HDU では，SIMPLE，NAXIS，EXTEND，PCOUNT，GCOUNT キーワードを省略できます.

(13) Binary Table HDU または Ascii Table HDU では，BITPIX，NAXIS，NAXIS2，PCOUNT，GCOUNT，TFIELDS キーワードを省略できます．Binary Table HDU では，NAXIS1 も省略できます.

(14) TTYPE$n$ 等のテーブルのカラム (フィールド) に関する記述では，キーワードの数字 $n$ のかわりに「#」記号を与える事により，自動的にカラム番号を割り当てます．この自動ナンバリングは，最初に現れた「#」記号付きキーワードをトリガとしてナンバリングのためのカウンタを増やす事により行われます.

(15) 「/」以降のコメントが省略された場合，FITS 標準のキーワードを持つレコードついては確実に SFITSIO 搭載のデフォルトのコメントで自動補完します．FITS 標準以外のキーワードについては，一般的なものについてのみ自動補完します[32].

(16) 「キーワード = 値 / コメント」の形式のレコードを作る場合は,「=」記号を省略する事はできません.

(17) 現在のバージョンでは，テンプレート上のキーワードの小文字・大文字は区別しませんが，将来の拡張を考え，キーワードは大文字で書いた方が良いでしょう.

[32] 詳細は APPENDIX2(§14) をご覧ください

# 9 CFITSIO 互換のローカル FITS 拡張

## 9.1 複数レコードにまたがる長いヘッダ値

SFITSIO では，CFITSIO と同様に，ヘッダの文字列値が 1 行のヘッダレコードに収まらない場合，次のように CONTINUE を使って保存します．

```
TELEM6  = 'CREON,SHTOP,FWPOSON,FWPOS_B1,FWPOS_B0,MPOSON,MPOS_B1,MPOS_B0,&'
CONTINUE  'RSTWIDELON,RSTWIDESON,RSTN170ON,RSTN60ON,LWBOOSTON,SWBOOSTON,&'
CONTINUE  'LWBIASON,SWBIASON,CALALON,CALASON,CALBON,SINALON,SINASON&'
CONTINUE  '' / elements in STATUS
```

SFITSIO の場合は，ファイルからの読み書きの時にこの拡張のための処理，すなわち，ファイル・オブジェクト間のデータ変換を行ないます．SFITSIO のオブジェクトには，文字列長の制限はありませんから，CFITSIO のようなロング値専用の API は存在しません．つまり，ファイル上の CONTINUE レコードの有無をユーザが気にする必要は無いわけです．

## 9.2 バイナリテーブルの固定長文字列の配列

CFITSIO では，TFORM$n$ = '120A10' のような指定をするか，TFORM$n$ = '120A' かつ TDIM$n$ = '(10,12)' のように指定する事で，12 個の 10 文字の文字列を 1 つのカラムで扱えるようになります．これらの FITS 拡張は，SFITSIO でもサポートしています．

SFITSIO ではさらに，TFORM$n$ = '120A10' かつ TDIM$n$ = '(6,2)' のように指定する事で，10 文字の文字列を，$6 \times 2$ の配列として扱う事も可能です．この指定は，TFORM$n$ = '120A' かつ TDIM$n$ = '(10,6,2)' と書く事もできます．

## 9.3 チェックサムとデータサムの書き込み

SFITSIO は，各 HDU のヘッダに CHECKSUM または DATASUM キーワードが存在した場合，FITS ファイルを保存する時に CFITSIO 互換のチェックサムまたはデータサムを自動的に書き込みます．例を示します．

```
CHECKSUM= 'LNXALNX2LNX8LNX8'   / HDU checksum : 2012-01-16T13:34:03
DATASUM = '2155872383'         / data unit checksum : 2012-01-16T13:34:03
```

なお，このチェックサムとデータサムは，基本的には HDU のバイトデータすべてを 32 ビット整数とみなして足し算をしているだけです．したがって，このチェックサムはデータの同一性を保証するには不十分であると考えられます．そのような用途には，md5sum を使う事をお勧めします．

なお，現在のバージョンの SFITSIO にはこのチェックサムとデータサムをチェックする機能はありません．

# 10 SFITSIO のローカル FITS 拡張

この章で述べる FITS の拡張は，そのほとんどがバイナリテーブルに関するものです．特に §10.8以降の拡張は宇宙科学研究開発機構・宇宙科学研究所の「あかり」プロジェクトや L1TSD プロジェクトで生まれたもので，観測衛星の複雑なデータをわかりやすく格納し，効率的にプロセッシングできるよう，長期にわたる関係者の様々な議論を経て策定されたものです。

## 10.1 FMTTYPE，FTYPEVER による FITS のエラーチェックとバージョン管理

§1.2で紹介したように,「あかり」プロジェクトでは TSD ファイルを定義したわけですが，プロジェクトがある程度進捗した段階において，プロジェクトで開発したソフトウェアが TSD ファイルを読み込んだ時に，どのように FITS ファイルの妥当性のチェックを行ない，TSD のバージョンによる動作の切り替えを行なうか，という議論が行なわれました．

そこで考え出されたのが，FITS フォーマットの「名前」と「バージョン」とを，プライマリ HDU のヘッダの「FMTTYPE」,「FTYPEVER」にセットする，という取り決めです．キーワード FMTTYPE の値には，(可能な限り) 世界的に unique なデータフォーマットを規定する文字列，すなわち「プロジェクト名 装置名 ファイルの種類」のような文字列をセットします.「あかり」プロジェクトでは「ASTRO-F TSD FIS_LW」のように定めました．FTYPEVER は，そのデータフォーマットのバージョンを示し，整数値で指定します．

なお，FTYPEVER の値はマイナーバージョンを示すために，3 ケタ以上の数値を使う事をお勧めします．あるいは，20080101 のような日付を使う方法もあります．

SFITSIO は,「FMTTYPE」,「FTYPEVER」を専用の API でサポートします．

## 10.2 ヘッダキーワードの大文字・小文字の区別

現在の FITS 規約では，キーワードに小文字を使う事は許されていません．しかし，SFITSIO では FITS ヘッダのキーワードの大文字と小文字とは区別されます．例えば，次のようなヘッダレコードを作る事ができます．

```
FOO     = 123
Foo     = 456
```

## 10.3 ヘッダのロングキーワード (最大 54 文字)

現在の FITS 規約では，ヘッダのキーワードは 8 文字までに制限されており,「空白 8 文字のキーワード」以外の場合においては，ヘッダレコードの **9 文字目は必ず “=” か空白文字**が現れます．つまり，9 文字目がそれら以外の場合については定義されていないわけです．

この未定義の部分を「ヘッダレコードの **9 文字目に “=” でも空白文字でもない文字が現れた場合，それは 8 文字を越えるロングキーワードとする**」と定義し,「ロングキーワードは “=” と空白は含んではならない」と約束すれば，過去の FITS ファイルにおける互換性の問題を引き起こさずに，可読性の高いロングキーワードを使う事ができます．

この定義に従って，SFITSIO では次のように単純にロングキーワードを保存します．

```
TTYPE12345= 'Mag     '          / column name
TFORM12345= '1D      '          / data format : 8-byte REAL
```

キーワードは最大 54 文字まで[33] で,「=」や空白文字を使う事はできません.

なお、ESO のコンベンションでは，ロングキーワードのヘッダレコードはそのレコードの先頭に HIERARCH をつけて区別していますが，これはスジが悪いと考えます.

## 10.4 999 を越えるアスキーテーブル・バイナリテーブルのカラム数

現在の FITS 規約におけるヘッダキーワードの長さ制限 (8 文字) により，アスキーテーブル，バイナリテーブルのカラムは 999 個までしか定義できません。

SFITSIO では、前項のヘッダのロングキーワード拡張により，アスキーテーブルまたはバイナリテーブルのカラム数は事実上無制限となっています.

## 10.5 コメント文字列に対する CONTINUE キーワードの適用

COMMENT レコードは長い文章を FITS ヘッダに入れられるので便利ですが,「このコメントはどこのヘッダレコードについての記述か」をライブラリは判断できません．これは，自動化の妨げになるので，コメント文はできるだけ通常のヘッダレコードの「/」の後に書くべきです.

一方,「/」の後のコメント文は，文字列値が長い場合には，単純なアルゴリズムでヘッダをフォーマットすると，コメント後部がカットされてしまうという問題があります.

この問題を解決するため，SFITSIO ではできる限りコメント文をすべて残すように，次のようにロング文字列拡張を使って文字列の最後で次の CONTINUE に接続するようにし，十分なコメント領域を確保するようにしています.

```
TELEM34 = 'BAD_FRAME,UNDEF_ANOM_FRAME,BLANK,IN_SAA,NEAR_MOON,UNTRUSTED_FRAME&'
CONTINUE  '' / element names
```

さらに，上記のようにしても切れてしまう場合は，コメント文も「&」と CONTINUE キーワードで保存するようにしています．例を示します.

```
MESSAGE = 'FITS (Flexible Image Transport System) format is defined in &'
CONTINUE  '"Astronomy and Astrophysics", volume 376, page 359; bibcode: &'
CONTINUE  '2001A&A...376..359H' / In SFITSIO, this message is not written &
CONTINUE  automatically. Therefore, SFITSIO is not CFITSIO :-)
```

この拡張は，ISAS/JAXA の L1TSD プロジェクトで考案されました.

## 10.6 ヘッダの文字列値における改行文字の定義

バイナリテーブルやアスキーテーブルのカラムの説明などは，HTML，VOTable，LaTeX 等の形式に変換しなければならない事があります．このような場合の自動化処理を考えると，カラムの説明などは，COMMENT にではなく，キーワードに対する文字列値として保存すべきです．この場合，様々なフォーマットにおいて統一性を確保するためには，改行文字の定義が必要です.

SFITSIO では，ヘッダの文字列値での “\n”(C 言語では"\\n") を改行文字とみなし，ヘッダレコードを整形する場合に，改行文字でレコードを分離します.

[33] 54 文字という長さは，値に最大・最小の 64 ビット整数が記述でき，かつ 1 文字のコメント文が書けるという条件から決まります.

```
TDESC3  = 'Trigger type flag,\n&'
CONTINUE  '  b1000000:SUD(trigd by SuperUpper Discriminator),\n&'
CONTINUE  '  b0100000:ANODE,\n&'
CONTINUE  '  b0010000:PIN0 b0001000:PIN1 b0000100:PIN2,\n&'
CONTINUE  '  b0000010:PIN3 b0000001:PSEUDO' / description of column
```

この拡張は，ISAS/JAXA の L1TSD プロジェクトで考案されました.

## 10.7 アスキーまたはバイナリテーブルのカラムに関する新規キーワードの宣言

テーブルのカラム (フィールド) に関するキーワードを独自に定義する事は，様々な機関においてよく行なわれている事です．代表的なものとしては，TLMIN$n$，TLMAX$n$ が挙げられます.

しかし，FITS の I/O ライブラリでは，これらがテーブルのカラム (フィールド) に属するキーワードなのか，そうではないのかは判定できません．したがって，FITS の I/O ライブラリでテーブルのカラムを削除したり移動したりすると，新たに定義されたカラムに関するキーワードは取り残されてしまったり，必要なものがコピーされないといった問題が発生します．また，テーブルの情報を HTML，VOTable，LaTeX 等に完全自動で変換する事もできません.

そこで，“TXFLDKWD” というキーワードを定義し，新規に追加されたテーブルのカラムに関するキーワードを次のように csv 形式で宣言するというルールを定めます.

```
TXFLDKWD= 'TLMIN,TLMAX,TALAS,TELEM,TDESC' / extended field keywords
```

なお，“TXFLDKWD” は，TTYPE$n$ 等の定義の前に現れるべきです．SFITSIO では，TXFLDKWD の拡張に API レベルで対応しています.

この拡張は，ISAS/JAXA の L1TSD プロジェクトで考案されました.

## 10.8 アスキーまたはバイナリテーブルのカラム名の別名定義

次のように，キーワード TALAS$n$ を使って，カラム名の別名を定義できます.

```
TTYPE4  = 'QUATERNION'          / Quaternion at boresight
TALAS4  = 'AOCU_ADS_Q'          / aliases of column name
```

ここで定義された別名は，SFITSIO の API でも有効です．なお，複数の別名を定義する場合は，csv 形式で定義します.

この拡張は，ISAS/JAXA の「あかり」プロジェクトで考案されました.

## 10.9 バイナリテーブルのカラム中の要素名の定義

キーワード TELEM$n$ を使って，次のようにバイナリテーブルの配列型のカラム中の要素に対して，名前をつける事ができます.

```
TTYPE34 = 'FLAG    '            / Flag for detector condition
TELEM34 = 'BAD_FRAME,UNDEF_ANOM_FRAME,BLANK,IN_SAA,NEAR_MOON,UNTRUSTED_FRAME&'
CONTINUE  '' / element names
TFORM34 = '8X      '            / data format : BIT
```

各要素の名前は TELEM$n$ キーワードのレコードに csv 形式で定義します．すべての要素に対して名前をつける必要はなく，省略した場合は左詰めで要素名が定義されます．この例では，データフォーマットは「ビット型」ですが，他の型 (整数型や実数型など) でも同様に指定できます。

SFITSIO の API においては，dvalue()，assign() などのメンバ関数の引数に，これらの要素名を指定して値を読み書きする事ができます.

この拡張は，ISAS/JAXA の「あかり」プロジェクトで考案されました.

## 10.10 バイナリテーブルのカラム中の bit 個数の定義

カラムがビット型の配列の場合 (TFORM$n$ が'$m$X' の場合)，§10.9 の記法をさらに拡張し，TELEM$n$ の値に，C 言語の構造体のビットフィールドと同様の記法でそれぞれの要素のビット幅を定義できます (同じ要素名を複数個連続して指定する事でビット幅を定義する事もできます).

次に例を示します.

```
TTYPE36 = 'QUALITY '           / Quality for each pixel condition
TFORM36 = '4000X   '           / data format : BIT
TDIM36  = '(40,100)'
TELEM36 = 'QUAL_CV_PARAM:2,QUAL_RC_PARAM:2,QUAL_RC_CF:2,QUAL_DF_EQ:2,&'
CONTINUE  'QUAL_RP_DATA:2,QUAL_RP_PARAM:2,QUAL_RP_TABLE:2,QUAL_FF_PARAM:2,&'
CONTINUE  'QUAL_FF_CF:2,QUAL_GPGL_CORR:2,QUAL_MTGL_CORR:2,QUAL_TR_HIST:2,&'
CONTINUE  'QUAL_TR_PARAM:2,QUAL_DK_DATA:2,QUAL_DK_PARAM:2,QUAL_DK_TABLE:2,&'
CONTINUE  'QUAL_FX_CORR:2,QUAL_FX_PARAM:2,,,,' / element names
```

この場合，TTYPE36，TFORM36，TDIM36 までは FITS 標準の記法であり，テーブルの 1 つのセルに，横 40 ×縦 100(合計 4000 個) のビットが定義されています．TELEM36 では，横 40 個に対して 2 ビットずつ要素名を定義しています．すべての要素に対して名前をつける必要はなく，省略した場合は左詰めで要素名が定義されます．1 つの要素が持つビットの数は，要素名の後のコロン「:」の後に記載します。なお，要素名にコロンを使うことはできますが，将来的に互換性に関する問題が引き起こされる可能性があるため，使うべきではないでしょう。

もちろん，SFITSIO の API においては，定義されたビット数からなる整数 (32 ビットまで) を読み書きできます．API による値の読み書きは，FITS ファイル上のバイトデータをそのままメモリに保持した状態で行われるので，ビットフィールドの定義は，ファイルサイズを小さくできると同時に，メモリの節約にもなります.

この拡張は，ISAS/JAXA の「あかり」プロジェクトで考案されました.

# 11 サポートされない FITS 規格と制限

ランダムグループ構造は SFITSIO ではサポートされていません.

バイナリテーブルの複素数と可変長配列については，ファイルの読み書きとテンプレートの読み込みは可能ですが，テーブルのセルの読み書きのための十分な API が提供されていません．どうしても必要な場合は低レベル API を使う必要があります.

# 12 リファレンス

## 12.1 定数

namespace sli の中で，namespace FITS を定義し，FITS ファイルを扱うのに必要な定数を定義しています．ユーザのコードでは，値そのものを書かないようにしましょう．

HDU の種類を示すために，次に示す定数を使います．

| 型 | SFITSIO で使う定数名 | 値 |
|---|---|---|
| const int | FITS::IMAGE_HDU | 0 |
| const int | FITS::ASCII_TABLE_HDU | 1 |
| const int | FITS::BINARY_TABLE_HDU | 2 |

ユーザヘッダレコードの状態を示すために，次に示す定数を使います．

| 型 | SFITSIO で使う定数名 | 値 |
|---|---|---|
| const int | FITS::NULL_RECORD | 0 |
| const int | FITS::NORMAL_RECORD | 1 |
| const int | FITS::DESCRIPTION_RECORD | 2 |

データの種類を示すために，次に示す定数を使います．

| 型 | SFITSIO で使う定数名 | 値 | Image | Binary Table |
|---|---|---|---|---|
| const int | FITS::BIT_T | 88 | | ○ |
| const int | FITS::BYTE_T | 66 | ○ | ○ |
| const int | FITS::LOGICAL_T | 76 | | ○ |
| const int | FITS::BOOL_T | 76 | | ○ |
| const int | FITS::ASCII_T | 65 | | ○ |
| const int | FITS::STRING_T | 65 | | ○ |
| const int | FITS::SHORT_T | 73 | ○ | ○ |
| const int | FITS::LONG_T | 74 | ○ | ○ |
| const int | FITS::LONGLONG_T | 75 | ○ | ○ |
| const int | FITS::FLOAT_T | 69 | ○ | ○ |
| const int | FITS::DOUBLE_T | 68 | ○ | ○ |
| const int | FITS::COMPLEX_T | 67 | | ○ |
| const int | FITS::DOUBLECOMPLEX_T | 77 | | ○ |
| const int | FITS::LONGARRDESC_T | 80 | | ○ |
| const int | FITS::LLONGARRDESC_T | 81 | | ○ |

## 12.2 型

namespace sli の中で，namespace fits を定義し，FITS ファイルを扱うのに必要な型を定義しています．

FITS ファイルの生のデータにアクセスするために，次に示す型を利用します.

| SFITSIO で使う型 | 実際に使われる型 | Image | Binary Table |
|---|---|---|---|
| fits::bit_t | struct _bit_t | | ○ |
| fits::byte_t | uint8_t | ○ | ○ |
| fits::logical_t | uint8_t | | ○ |
| fits::short_t | int16_t | ○ | ○ |
| fits::long_t | int32_t | ○ | ○ |
| fits::longlong_t | int64_t | ○ | ○ |
| fits::float_t | float | ○ | ○ |
| fits::double_t | double | ○ | ○ |
| fits::complex_t | float _Complex | | ○ |
| fits::doublecomplex_t | double _Complex | | ○ |
| fits::ascii_t | char | | ○ |
| fits::longarrdesc_t | struct _longarrdesc_t | | ○ |
| fits::llongarrdesc_t | struct _llongarrdesc_t | | ○ |

FITS ヘッダを定義するために，次に示す構造体を利用します.

| SFITSIO で使う型 | 構造体の定義 |
|---|---|
| fits::header_def | struct {<br>const char *keyword;<br>const char *value;<br>const char *comment;<br>} |

アスキーテーブル・バイナリテーブルのカラムを定義するために，次に示す構造体を利用します.

| SFITSIO で使う型 | 構造体の定義 |
|---|---|
| fits::table_def | struct {<br>const char *ttype;<br>const char *ttype_comment;<br>const char *talas;<br>const char *telem;<br>const char *tunit;<br>const char *tunit_comment;<br>const char *tdisp;<br>const char *tform;<br>const char *tdim;<br>const char *tnull;<br>const char *tzero;<br>const char *tscal;<br>} |

## 12.3 FITS 全体に対する操作

ここでは，ストリーム入出力や，HDU の構成を操作するための API を解説します.

### 12.3.1 read_stream()

**NAME**

read_stream() — ストリームから読み込む

**SYNOPSIS**

```
ssize_t read_stream( const char *path );
ssize_t readf_stream( const char *path_fmt, ... );
ssize_t vreadf_stream( const char *path_fmt, va_list ap );
```

**DESCRIPTION**

path で指定されたファイルあるいは URL(file://, http://, ftp://に対応) から FITS ファイルを読み込み，内容すべてをオブジェクトに取り込みます．path のファイル名，http サーバから得た MIME ヘッダから判断して，必要な場合は zlib，bzlib 経由で読み込みを行ないます[34)]．ftp サーバから読み込む場合は，path に ftp://username:password@hostname/…の書式で，ユーザ名とパスワードを設定できます．ユーザ名とパスワードを設定しない場合は，匿名 (anonymous) でのアクセスとなります.

このメンバ関数を使う前に, hdus_to_read().assign(), cols_to_read().assign() (§12.3.2) を使うと，特定の HDU・アスキーテーブルまたはバイナリテーブルの特定のカラムだけを読み込む事ができます.

readf_stream() メンバ関数の場合，path_fmt 以降の引数を libc の printf() のそれと同様に指定します．printf() の書式については，§12.4.9 の解説を参照してください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | path | ファイル名 (URL 名) |
| [I] | path_fmt | ファイル名 (URL 名) のフォーマット指定 |
| [I] | ... | ファイル名 (URL 名) の各要素データ |
| [I] | ap | ファイル名 (URL 名) の全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

| | | |
|---|---|---|
| 非負の値 | : | 読み込んだストリームのバイトサイズ (圧縮ファイルの場合は展開後のサイズ). |
| 負の値 (エラー) | : | ファイルが見つからないなど，ストリームの読み込みに失敗した場合. |

**EXCEPTION**

指定された FITS ファイルに対してメモリ容量が十分でない場合，バッファの確保に失敗するような致命的な事態となった時に，sfitsio を構成するクラスと SLLIB が提供するクラスに由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.3.2の EXAMPLES を参照してください.

---

[34)] これは，SLLIB の digeststreamio クラスで行なっています．APPENDIX4(§16) に解説があります.

チュートリアルの §5.2, §5.3 にも使用例があります.

---

### 12.3.2 hdus_to_read().assign(), cols_to_read().assign()

**NAME**

hdus_to_read().assign(), cols_to_read().assign() — ストリームから読む HDU とカラムの指定

**SYNOPSIS**

```
tarray_tstring &hdus_to_read().assign( const char *hdu0, const char *hdu1, ... );
tarray_tstring &hdus_to_read().assign( const char *hdus[] );
tarray_tstring &cols_to_read().assign( const char *col0, const char *col1, ... );
tarray_tstring &cols_to_read().assign( const char *cols[] );
```

**DESCRIPTION**

これらのメンバ関数を，read_stream() メンバ関数 (§12.3.1) の前に使う事で，特定の HDU・アスキーまたはバイナリテーブルの特定のカラムだけを読み込む事ができます (ただし，Primary HDU だけは読み込みを省略する事はできません). 引数には，HDU 名，アスキーまたはバイナリテーブルのカラム名を列挙し，NULL で終端するようにします.

すべての HDU, あるいはすべてのカラムが必要な場合は，それぞれ，`hdus_to_read().init()`, `cols_to_read().init()` を使います.

これらのメンバ関数でセットされた値は，init() メンバ関数 (§12.3.15) で消去されます.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `hdu0,hdu1...` | HDU 名 (終端は NULL を指定) |
| [I] | `hdus[]` | HDU 名の配列 (終端は NULL を指定) |
| [I] | `col0,col1...` | バイナリまたはアスキーテーブルのカラム名 (終端は NULL を指定) |
| [I] | `cols[]` | バイナリまたはアスキーテーブルのカラム名の配列 (終端は NULL を指定) |

([I]：入力，[O]：出力)

**RETURN VALUE**

設定された HDU 名またはカラム名を含む `tarray_tstring` オブジェクトへの参照を返します.

**EXCEPTION**

内部バッファの確保に失敗した場合，SLLIB が提供するクラスに由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードは，Primary HDU と FIS_OBS HDU とを読み込み，バイナリテーブルについては，“AFTIME”，“DET”，“RA”，“DEC” という名のカラムだけ読み込みます.

```
fitscc fits;
/* Required HDUs */
fits.hdus_to_read().assign("Primary","FIS_OBS",NULL);
/* Required columns in binary tables */
fits.cols_to_read().assign("AFTIME","DET","RA","DEC",NULL);
```

```
/* Reading a file */
r_size = fits.readf_stream("my_file_no.%d.fits.gz",i);
```

---

### 12.3.3 write_stream()

**NAME**

write_stream() — ストリームへ書き出す

**SYNOPSIS**

```
ssize_t write_stream( const char *path );
ssize_t writef_stream( const char *path_fmt, ... );
ssize_t vwritef_stream( const char *path_fmt, va_list ap );
```

**DESCRIPTION**

オブジェクトの内容すべてを path で指定されたファイルへ書き出します．path のファイル名から判断して，必要な場合は zlib，bzlib 経由で書き込みを行ないます[35]．ftp サーバへ書き込む場合は，path に ftp://username:password@hostname/…の書式で，ユーザ名とパスワードを設定できます．ユーザ名とパスワードを設定しない場合は，匿名 (anonymous) でアクセスします．

writef_stream() メンバ関数の場合，path_fmt 以降の引数を libc の printf() のそれと同様に指定します．

**PARAMETER**

| | | |
|---|---|---|
| [I] | path | ファイル名 (URL 名) |
| [I] | path_fmt | ファイル名 (URL 名) のフォーマット指定 |
| [I] | ... | ファイル名 (URL 名) の各要素データ |
| [I] | ap | ファイル名 (URL 名) の全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

| | | |
|---|---|---|
| 非負の値 | : | 書き込んだストリームのバイトサイズ (圧縮ファイルの場合は圧縮前のサイズ)． |
| 負の値 (エラー) | : | パーミッションに問題がある場合などの理由で，ストリームの書き込みに失敗した場合． |

**EXCEPTION**

メモリバッファの操作若しくは出力ファイルの操作に失敗した場合 (例えば，出力ファイルへの書出し時の予期せぬエラー等)，SLLIB に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

次のコードは，例えば i が 1 の場合，ファイル my_file_no.1.fits.bz2 に fits オブジェクトの内容をすべて書き出します．サフィックスが ".bz2" なので，この場合は bzip2 で圧縮してファイルに書き込まれます．

```
fitscc fits;
w_size = fits.writef_stream("my_file_no.%d.fits.bz2",i);
```

[35] これは，SLLIB の digeststreamio クラスで行なっています．APPENDIX4(§16) に解説があります．

チュートリアルの §5.2，§5.3 にも使用例があります.

---

### 12.3.4 access_stream()

**NAME**

access_stream() — ストリームへのアクセス

**SYNOPSIS**

```
ssize_t access_stream( const char *path );
ssize_t accessf_stream( const char *path_fmt, ... );
ssize_t vaccessf_stream( const char *path_fmt, va_list ap );
```

**DESCRIPTION**

これらのメンバ関数の引数は，スクリプト言語 Perl の `open()` のそれに似ており，`path` あるいは `path_fmt` がファイルや URL を示している場合はそれに対して読み込みか書き込みを行ないます．また，コマンドを示している場合はそれを実行し，パイプで接続して，パイプ経由で読み込みか書き込みを行ないます[36].

読み込みか書き込みかは，`path` あるいは `path_fmt` がファイルや URL を示している場合は，`"< infile.fits"` や `"> outfile.fits"` のように「<」あるいは「>」をつけて表現します.「<」がある場合は読み込みを行ない (EXAMPLE-1 参照),「>」がある場合は書き込みを行ないます.「<」「>」の指定が無い場合は，読み込みを行ないます．圧縮されたファイル (gzip か bzip2 形式) が指定された場合，自動的に圧縮・伸長します.

`path` あるいは `path_fmt` がコマンドを示している場合，`"command |"` や `"| command"` のように,「|」を `path` の最後か先頭につけて表現します．「|」が `path` の最後につけられている場合は，指定したコマンドを実行してパイプで接続し，そのコマンドの標準出力からの出力を FITS ファイルの内容として読み込みます.「|」が `path` の先頭につけられている場合は，指定したコマンドを実行してパイプで接続し，このコマンドに対して FITS ファイルの内容を書き出します (EXAMPLE-2 参照)．`path` あるいは `path_fmt` がコマンドの場合,「`/bin/sh -c` コマンド」の形で実行しますので，`path` にはパイプやリダイレクトの記号 (|,<,>) を含む事ができます (EXAMPLE-2 参照).

accessf_stream() メンバ関数の場合，`path_fmt` 以降の引数を libc の `printf()` のそれと同様に指定します.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `path` | ファイル名 (URL 名) またはコマンド |
| [I] | `path_fmt` | ファイル名 (URL 名) またはコマンドのフォーマット指定 |
| [I] | `...` | ファイル名 (URL 名) またはコマンドの各要素データ |
| [I] | `ap` | ファイル名 (URL 名) またはコマンドの全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

---

[36] これは，SLLIB の digeststreamio クラスで行なっています．APPENDIX4(§16) に解説があります.

| | | |
|---|---|---|
| 非負の値 | : | 読み書きしたストリームのバイトサイズ (圧縮ファイルの場合は圧縮前のサイズ). |
| 負の値 (エラー) | : | ファイルが見つからないなど，ストリームの読み込みに失敗した場合．パーミッションに問題がある場合などの理由で，ストリームの書き込みに失敗した場合. |

**EXCEPTION**

メモリバッファの操作若しくは出力ファイルの操作に失敗した場合 (例えば，出力ファイルへの書出し時の予期せぬエラー等)，SLLIB に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLE-1**

次のコードは，ファイル `my_file.fits.bz2` を読み込み，その内容をすべて fits オブジェクトに取り込みます．サフィックスが ".bz2" なので，この場合は bzip2 で伸長してファイルを読み込みます.

```
fitscc fits;
r_size = fits.access_stream("< my_file.fits.bz2");
```

**EXAMPLE-2**

次のコードは，HTTP over SSL でサーバに接続し，FITS ファイルを読む例です.

```
fitscc fits;
r_size = fits.accessf_stream("wget -O - %s | gzip -dc |",
                             "https://foo/secret.fits.gz");
```

チュートリアルの §5.4 では，マルチスレッド対応の圧縮ツールを利用した例を紹介しています.

---

### 12.3.5 read_template()

**NAME**

read_template() — FITS テンプレートファイルの読み込む

**SYNOPSIS**

```
ssize_t read_template( int flags, const char *path );
ssize_t vreadf_template( int flags, const char *path_fmt, va_list ap );
ssize_t readf_template( int flags, const char *path_fmt, ... );
```

**DESCRIPTION**

`path` で指定されたパスあるいは URL(`file://`，`http://`，`ftp://`に対応) から FITS テンプレート (SFITSIO の FITS テンプレートについては，§8を参照) を読み込み，その内容に従って fitscc オブジェクトを新規に作成します．`path` のファイル名，http サーバから得た MIME ヘッダから判断して，必要な場合は zlib，bzlib 経由で読み込みを行ないます．ftp サーバから読み込む場合は，`path` に `ftp://username:password@hostname/`…の書式で，ユーザ名とパスワードを設定できます．ユーザ名とパスワードを設定しない場合は，匿名 (anonymous) でのアクセスとなります.

readf_template() メンバ関数の場合，`path_fmt` 以降の引数を libc の `printf()` のそれと同様に指定します．`printf()` の書式については，§12.4.9 の解説を参照してください.

**PARAMETER**

[I] `flags` 動作切り替えのためのフラグ (現在は未使用; 常に 0 を指定)
[I] `path` ファイル名 (URL 名)
[I] `path_fmt` ファイル名 (URL 名) のフォーマット指定
[I] `...` ファイル名 (URL 名) の各要素データ
[I] `ap` ファイル名 (URL 名) の全要素データ
([I]：入力，[O]：出力)

**RETURN VALUE**

0 : 正常終了.
負の値 (エラー) : ファイルが見つからないなど，ストリームの読み込みに失敗した場合.

**EXCEPTION**

指定された FITS テンプレートに対してメモリ容量が十分でない場合，バッファの確保に失敗するような致命的な事態となった時に，sfitsio を構成するクラスと SLLIB が提供するクラスに由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fitscc fits;
status = fits.read_template(0, "template/image1.tpl");
```

テンプレートの内容については，§8を参照してください.

---

### 12.3.6 stream_length()

**NAME**

stream_length() — ストリームへ書き出されるサイズを返す

**SYNOPSIS**

```
ssize_t stream_length();
```

**DESCRIPTION**

システムヘッダを再編し，write_stream() で書き出されるファイルサイズを返します.

**RETURN VALUE**

非負の値 : ストリームへ書き出されるサイズ.
負の値 (エラー) : エラーが発生した場合 (debug モードのみ).

**EXCEPTION**

メモリ不足により文字列処理に失敗した場合，SLLIB に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fitscc fits;
w_size = fits.stream_length();
```

---

### 12.3.7 length()

**NAME**

length() — HDU の個数

**SYNOPSIS**

```
long length() const;
```

**RETURN VALUE**

HDU の個数を返します.

**EXAMPLES**

```
long hdu_count = fits.length();
```

チュートリアルの §5.2 にも使用例があります.

---

### 12.3.8 fmttype()

**NAME**

fmttype() — フォーマットタイプ名

**SYNOPSIS**

```
const char *fmttype() const;
```

**DESCRIPTION**

フォーマットタイプ名を返します. 設定されていない場合は, NULL を返します.

返される値はオブジェクト内部バッファのアドレスですから, オブジェクトが破棄されたり, 名称が変更された場合は無効になります.

フォーマットタイプ名についての詳細は, §10.1 を参照してください.

**RETURN VALUE**

フォーマットタイプ名のアドレスを返します.

**EXAMPLES**

```
printf("Format Type = %s\n", fits.fmttype());
```

---

### 12.3.9 ftypever()

**NAME**

ftypever() — フォーマットタイプのバージョン番号

**SYNOPSIS**

```
long long ftypever() const;
```

**RETURN VALUE**

フォーマットタイプのバージョン番号を返します.

**EXAMPLES**

```
printf("Verson of Format Type = %lld\n", fits.ftypever());
```

---

### 12.3.10 hduname(), extname()

**NAME**

hduname(), extname() — HDU 名の取得

**SYNOPSIS**

```
const char *hduname( long index ) const;
const char *extname( long index ) const;
```

**DESCRIPTION**

index で指定された HDU の名称を返します．設定されていない場合は，NULL を返します．

返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，名称が変更された場合は無効になります．

**PARAMETER**

[I] index HDU のインデックス
([I]：入力，[O]：出力)

**RETURN VALUE**

HDU 名のアドレスを返します．

**EXAMPLES**

次のコードは，全ての HDU 名を標準出力に列挙します．

```
long i;
for ( i=0 ; i < fits.length() ; i++ ) {
    printf("HDU[%ld] Name is %s\n", i, fits.hduname(i));
}
```

チュートリアルの §5.2 にも使用例があります．

---

### 12.3.11 hduver(), extver()

**NAME**

hduver(), extver() — HDU のバージョン

**SYNOPSIS**

```
long long hduver( long index ) const;
long long hduver( const char *name ) const;
long long extver( long index ) const;
long long extver( const char *name ) const;
```

**DESCRIPTION**

index または name で指定された HDU のバージョンを返します．

**PARAMETER**

[I] index HDU のインデックス
[I] name HDU 名
([I]：入力，[O]：出力)

**RETURN VALUE**

HDU のバージョンを返します.

**EXAMPLES**

次のコードは，全ての HDU のバージョンを標準出力に列挙します.

```
long i;
for ( i=0 ; i < fits.length() ; i++ ) {
    printf("HDU[%ld] Version is %lld\n", i, fits.hduver(i));
}
```

---

### 12.3.12 hdulevel(), extlevel()

**NAME**

hdulevel(), extlevel() — HDU のレベル

**SYNOPSIS**

```
long long hdulevel( long index ) const;
long long hdulevel( const char *name ) const;
long long extlevel( long index ) const;
long long extlevel( const char *name ) const;
```

**DESCRIPTION**

index または name で指定された HDU のレベル (ヘッダの EXTLEVEL の値) を返します.

**PARAMETER**

[I] index HDU のインデックス
[I] name HDU 名
([I]：入力，[O]：出力)

---

### 12.3.13 hdutype(), exttype()

**NAME**

hdutype(), exttype() — HDU のタイプ

**SYNOPSIS**

```
int hdutype( long index ) const;
int hdutype( const char *name ) const;
int exttype( long index ) const;
int exttype( const char *name ) const;
```

**DESCRIPTION**

index または name で指定された HDU のタイプを返します.
返る値は，FITS::IMAGE_HDU，FITS::ASCII_TABLE_HDU，FITS::BINARY_TABLE_HDU のいずれかです.

**PARAMETER**

[I] `index` HDU のインデックス
[I] `name` HDU 名
([I]：入力，[O]：出力)

**RETURN VALUE**

HDU のタイプを返します.

**EXAMPLES**

```
switch ( fits.hdutype(index) ) {
  case FITS::IMAGE_HDU:
      printf("イメージ HDU です!\n");
      break;
  case FITS::ASCII_TABLE_HDU:
      printf("ASCII テーブル HDU です!\n");
      break;
  case FITS::BINARY_TABLE_HDU:
      printf("バイナリテーブル HDU です!\n");
      break;
  default:
      printf("不明なタイプの HDU です!\n");
      break;
}
```

---

### 12.3.14 index()

**NAME**

index() — HDU 番号

**SYNOPSIS**

```
long index( const char *name ) const;
long indexf( const char *name_fmt, ... ) const;
long vindexf( const char *name_fmt, va_list ap ) const;
```

**DESCRIPTION**

HDU 名 `name` の番号を返します.

`"Primary"`だけは特別で，ヘッダキーワード `EXTNAME` が設定されていない場合でも，プライマリ HDU が存在すれば，常に 0 番を返します.

見つからない場合は，負の値を返します.

`name_fmt` 以降の引数は，libc の `printf()` のそれと同様に指定可能です.

**PARAMETER**

[I] `name` HDU 名
[I] `name_fmt` HDU 名のフォーマット文字列
[I] `...` HDU 名の全要素データ
[I] `ap` HDU 名の全要素データ
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : HDU 名 name の番号.
負の値 (エラー) : 見つからない場合.

**EXCEPTION**

フォーマット文字列変換時のメモリバッファ操作に失敗した場合, SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，指定した HDU 名を持つ HDU のインデックス番号を取得します.

```
long index;
index = fits.index("FIS_OBS");
```

---

### 12.3.15 init()

**NAME**

init() — オブジェクトの初期化

**SYNOPSIS**

```
fitscc &init();
fitscc &init( const fitscc &obj );
```

**DESCRIPTION**

オブジェクトの内容すべて破棄し，初期化します. 引数 obj が指定されている場合は，オブジェクト obj の内容をすべて自身にコピーします.

**PARAMETER**

[I] obj 初期化後にコピー元となる fitscc オブジェクト
([I] : 入力，[O] : 出力)

**RETURN VALUE**

自身の参照を返します.

**EXCEPTION**

内部メモリバッファ操作に失敗した場合 (例えば，空きメモリ領域に対して引数 obj の持つデータが大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
/* Initialize Fits Object!! */
fits.init();
```

---

### 12.3.16 append_image()

**NAME**

append_image() — Image HDU の追加

**SYNOPSIS**

```
fitscc &append_image( const char *hduname, long long hduver,
                      int type, long naxis0, long naxis1 = 0, long naxis2 = 0 );
fitscc &append_image( const char *hduname, long long hduver,
                      int type, long naxisx[], long ndim );
fitscc &append_image( const char *hduname, long long hduver,
                      const fits_image &src );
fitscc &append_image( const fits_image &src );
```

**DESCRIPTION**

Image HDU を追加します．引数 hduname は，HDU の名前を指定し，hduver はそのバージョンを指定します．これらの値は FITS ヘッダの EXTNAME，EXTVER に反映されます．引数 hduname に NULL を指定すると，この機能を無効にできますが，NULL の指定は SFITSIO を使う上ではお勧めできません．

type は，FITS::DOUBLE_T, FITS::FLOAT_T, FITS::LONGLONG_T, FITS::LONG_T, FITS::SHORT_T, FITS::BYTE_T のいずれかを指定します．

naxis0，naxis1，naxis2 はそれぞれ，軸 x, y, z のピクセル数を指定します．naxis1，naxis2 は省略可能で，naxis0 だけを指定した時は一次元，naxis1 までを指定した時は二次元となります．三次元を越える場合は，naxisx，ndim を指定します．

**PARAMETER**

| | | |
|---|---|---|
| [I] | hduname | HDU 名 |
| [I] | hduver | HDU バージョン |
| [I] | type | HDU のタイプ |
| [I] | naxis0 | X 軸のピクセル数 |
| [I] | naxis1 | Y 軸のピクセル数 |
| [I] | naxis2 | Z 軸のピクセル数 |
| [I] | naxisx | 各次元軸のピクセル数リスト |
| [I] | ndim | naxisx の要素数 |
| [I] | src | 追加するイメージオブジェクト |

([I]：入力，[O]：出力)

**RETURN VALUE**

自身の参照を返します．

**EXCEPTION**

内部メモリバッファ操作に失敗した場合 (例えば，空きメモリ領域に対して追加する画像データが大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

次のコードは，HDU 名 “X-BAND”，HDU バージョン 100，double 型で 1024 × 1024 のイメージ HDU をオブジェクト fits に追加します．

```
fits.append_image("X-BAND", 100,  FITS::DOUBLE_T, 1024, 1024);
```

チュートリアルの §5.9 にも使用例があります．

### 12.3.17 append_table()

**NAME**

append_table() — Ascii Table HDU または Binary Table HDU の追加

**SYNOPSIS**

```
fitscc &append_table( const char *hduname, long long hduver,
                      const fits::table_def defs[], bool ascii = false );
fitscc &append_table( const char *hduname, long long hduver,
                      const fits_table &src );
fitscc &append_table( const fits_table &src );
```

**DESCRIPTION**

Ascii Talbe HDU または Binary Table HDU を追加します．引数 ascii が true の場合は，Ascii Talbe HDU を追加します．引数 hduname は，HDU の名前を指定し，hduver はそのバージョンを指定します．これらの値は FITS ヘッダの EXTNAME，EXTVER に反映されます．引数 hduname に NULL を指定すると，この機能を無効にできますが，NULL の指定は SFITSIO を使う上ではお勧めできません．

defs で，アスキーテーブルまたはバイナリテーブルの定義を与えます．fits::table_def 構造体のメンバは次のようになっています．

```
typedef struct {
    const char *ttype;          /* カラム名 */
    const char *ttype_comment;
    const char *const *talas;   /* カラム名の別名 */
    const char *const *telem;   /* 要素名 */
    const char *tunit;          /* 物理単位 */
    const char *tunit_comment;
    const char *tdisp;          /* ディスプレイフォーマット */
    const char *tform;          /* カラムの型 */
    const char *tdim;           /* 配列の指定 */
    const char *tnull;          /* ブランクの値 */
    const char *tzero;          /* ゼロ点 */
    const char *tscal;          /* スケーリングファクター */
    const char *theap;          /* (未サポート) */
} fits::table_def;
```

この構造体の定義はバイナリテーブルのヘッダのキーワード名をそのまま使っている事からも，バイナリテーブルの場合は，それぞれのメンバに TTYPE$n$，TFORM$n$ 等の値をそのまま代入すれば良いようになっています．

一方，アスキーテーブルの場合は，この構造体のメンバ名はヘッダのキーワード名と一対一になっていないので注意してください．まず，tform には，必ず数字+「A」の形でカラムの文字列の長さを指定します．例えば，"16A"のように指定します (ただし，"120A10"のような指定はできません)．また，telem，tdim の指定は無意味です．tdisp には，アスキーテーブルの TFORM$n$ の指定を与え，これは SFITSIO のメンバ関数の引数の値 (数値または文字列) をアスキーテーブルに書き込む時に，引数の値から文字列に変換するフォーマットとして利用されます．指定できる形式は，A$w$，I$w$，F$w.d$，E$w.d$，D$w.d$ のいずれかです．

指定が不要な項目は，NULL か""を代入します．なお，THEAP は SFITSIO ではサポートしていません．配列 defs の最後では，すべてのメンバが NULL になっている必要があります．

**PARAMETER**

[I] hduname HDU 名
[I] hduver HDU バージョン
[I] defs fits::table_def 構造体
[I] ascii 追加する Table HDU の種類 (false:Binary true:Ascii)
[I] src 追加するテーブルオブジェクト
([I]：入力，[O]：出力)

**RETURN VALUE**

自身の参照を返します．

**EXCEPTION**

内部メモリバッファ操作に失敗した場合 (例えば，空きメモリ領域に対して追加するテーブルのデータが大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

次のコードは，HDU 名 “EVENT”，HDU バージョン 100 で，構造体配列 def の指定に従ったバイナリテーブル HDU をオブジェクト fits に追加します．

```
const fits::table_def def[] = {
    // TTYPE,comment, talas, telem, TUNIT,comment, TDISP, TFORM, TDIM
    { "TIME","satellite time", NULL, NULL, "s","", "D16.3", "1D", "" },
    { "NAME","", NULL, NULL, "","", "", "128A16", "(4,2)" },
    { NULL }
};
fits.append_table("EVENT", 100, defs);
```

チュートリアルの §5.15，§5.16 にも使用例があります．

---

### 12.3.18 insert_image()

**NAME**

insert_image() — Image HDU の挿入

**SYNOPSIS**

```
fitscc &insert_image( long index0,
                      const char *hduname, long long hduver,
                      int type, long naxis0, long naxis1 = 0, long naxis2 = 0 );
fitscc &insert_image( long index0,
                      const char *hduname, long long hduver,
                      int type, long naxisx[], long ndim );
fitscc &insert_image( const char *hduname0,
                      const char *hduname, long long hduver,
```

```
                        int type, long naxis0, long naxis1 = 0, long naxis2 = 0 );
fitscc &insert_image( const char *hduname0,
                        const char *hduname, long long hduver,
                        int type, long naxisx[], long ndim );
fitscc &insert_image( long index0,
                        const char *hduname, long long hduver,
                        const fits_image &src );
fitscc &insert_image( const char *hduname0,
                        const char *hduname, long long hduver,
                        const fits_image &src );
fitscc &insert_image( long index0, const fits_image &src );
fitscc &insert_image( const char *hduname0, const fits_image &src );
```

**DESCRIPTION**

index0 または hduname0 で指定された HDU に Image HDU を挿入します.

hduname 以降の引数の仕様は，append_image()(§12.3.16) の場合と同様です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | index0 | 挿入位置を示す HDU インデックス |
| [I] | hduname0 | 挿入位置を示す HDU 名 |
| [I] | hduname | HDU 名 |
| [I] | hduver | HDU バージョン |
| [I] | type | HDU のタイプ |
| [I] | naxis0 | X 軸のピクセル数 |
| [I] | naxis1 | Y 軸のピクセル数 |
| [I] | naxis2 | Z 軸のピクセル数 |
| [I] | naxisx | 各次元軸のピクセル数リスト |
| [I] | ndim | naxisx の要素数 |
| [I] | src | 挿入するイメージオブジェクト |

([I]：入力，[O]：出力)

**RETURN VALUE**

自身の参照を返します.

**EXCEPTION**

内部メモリバッファ操作に失敗した場合 (例えば，空きメモリ領域に対して挿入する画像データが大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，HDU 名 “X0-BAND”，HDU バージョン 100，double 型の 1024 × 1024 のイメージ HDU を，HDU“X-BAND” の前に挿入します.

```
fits.insert_image("X-BAND",  "X0-BAND", 100, FITS::DOUBLE_T, 1024, 1024);
```

### 12.3.19 insert_table()

**NAME**

insert_table() — Ascii Table HDU または Binary Table HDU の挿入

**SYNOPSIS**

```
fitscc &insert_table( long index0,
                      const char *hduname, long long hduver,
                      const fits::table_def defs[], bool ascii = false );
fitscc &insert_table( const char *hduname0,
                      const char *hduname, long long hduver,
                      const fits::table_def defs[], bool ascii = false );
fitscc &insert_table( long index0,
                      const char *hduname, long long hduver,
                      const fits_table &src );
fitscc &insert_table( const char *hduname0,
                      const char *hduname, long long hduver,
                      const fits_table &src );
fitscc &insert_table( long index0, const fits_table &src );
fitscc &insert_table( const char *hduname0, const fits_table &src );
```

**DESCRIPTION**

index0 または hduname0 で指定された HDU に Ascii Table HDU または Binary Table HDU を挿入します.

ただし，Primary HDU に挿入する事はできません.

hduname 以降の引数の仕様は，append_table()(§12.3.17) の場合と同様です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | index0 | 挿入位置を示す HDU インデックス |
| [I] | hduname0 | 挿入位置を示す HDU 名 |
| [I] | hduname | HDU 名 |
| [I] | hduver | HDU バージョン |
| [I] | defs | fits::table_def 構造体 |
| [I] | ascii | 挿入する Table HDU の種類 (false:Binary true:Ascii) |
| [I] | src | 挿入するテーブルオブジェクト |

([I]：入力，[O]：出力)

**RETURN VALUE**

自身の参照を返します.

**EXCEPTION**

内部メモリバッファ操作に失敗した場合 (例えば，空きメモリ領域に対して挿入するテーブルのデータが大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，HDU 名 “X-TABLE”，HDU バージョン 100，構造体配列 defs で定義されたテー

ブル HDU を，HDU“X-BAND” の前に挿入します (構造体配列 defs については，§12.3.17の EXAMPLES を参照してください).

```
fits.insert_table("X-BAND",  "X-TABLE", 100, defs);
```

---

### 12.3.20 erase()

**NAME**

erase() — HDU の削除

**SYNOPSIS**

```
fitscc &erase( long index );
fitscc &erase( const char *hduname );
```

**DESCRIPTION**

`index` または `hduname` で指定された HDU を削除します.

ただし，Primary HDU の次の HDU が Image HDU ではない場合，Primary HDU を削除する事はできません.

**PARAMETER**

[I] `index` 消去される HDU のインデックス
[I] `hduname` 消去される HDU の名称
([I]：入力，[O]：出力)

**RETURN VALUE**

自身の参照を返します.

**EXCEPTION**

内部メモリバッファ操作に失敗した場合, SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.erase("X-BAND");
```

---

### 12.3.21 assign_fmttype()

**NAME**

assign_fmttype() — フォーマットタイプ名の変更.

**SYNOPSIS**

```
fitscc &assign_fmttype( const char *fmttype, long long ftypever );
```

**DESCRIPTION**

フォーマットタイプの名称を変更します．引数 `fmttype` は，世界的に unique なデータフォーマットを規定する文字列で，`ftypever` はそのバージョンを指定します．これらの値は，プライマリ HDU のヘッダ `FMTTYPE`，`FTYPEVER` に反映されます.

fmttype，ftypever は，ファイルから FITS を読み込んだ時の，FITS の妥当性チェックに利用できます.

フォーマットタイプ名についての詳細は，§10.1 を参照してください.

**PARAMETER**

[I] fmttype セットするフォーマットタイプ名
[I] ftypever セットするフォーマットタイプのバージョン
([I]：入力，[O]：出力)

**EXCEPTION**

内部メモリバッファ操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**RETURN VALUE**

自身の参照を返します.

**EXAMPLES**

```
fits.assign_fmttype("ASTRO-X ALL-SKY SURVEY IMAGE", 101);
```

チュートリアルの §5.9，§5.15，§5.16 にも使用例があります.

---

### 12.3.22 assign_ftypever()

**NAME**

assign_ftypever() — フォーマットタイプバージョンの変更.

**SYNOPSIS**

```
fitscc &assign_ftypever( long long ftypever );
```

**DESCRIPTION**

フォーマットタイプのバージョンを変更します．値は，プライマリ HDU のヘッダ FTYPEVER に反映されます.

fmttype，ftypever は，ファイルから FITS を読み込んだ時の，FITS の妥当性チェックに利用できます.

**PARAMETER**

[I] ftypever セットするフォーマットタイプのバージョン
([I]：入力，[O]：出力)

**RETURN VALUE**

自身の参照を返します.

**EXAMPLES**

```
fits.assign_ftypever(102);
```

---

### 12.3.23 assign_hduname(), assign_extname()

**NAME**

assign_hduname(), assign_extname() — HDU 名の変更.

**SYNOPSIS**

```
fitscc &assign_hduname( long index, const char *name );
fitscc &assign_extname( long index, const char *name );
```

**DESCRIPTION**

index で指定された HDU の名称を変更します．名称 name はヘッダ EXTNAME に反映されます.

Primary HDU の場合も指定可能です.

**PARAMETER**

[I] index 変更対象を示す HDU インデックス
[I] name セットする HDU 名
([I]：入力，[O]：出力)

**EXCEPTION**

内部メモリバッファ操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**RETURN VALUE**

自身の参照を返します.

**EXAMPLES**

```
fits.assign_hduname("X-BAND");
```

---

### 12.3.24 assign_hduver(), assign_extver()

**NAME**

assign_hduver() — HDU のバージョン番号の変更.

**SYNOPSIS**

```
fitscc &assign_hduver( long index, long long ver );
fitscc &assign_extver( long index, long long ver );
```

**DESCRIPTION**

index で指定された HDU のバージョン番号を変更します．ver はヘッダ EXTVER に反映されます.

Primary HDU の場合も指定可能です.

**PARAMETER**

[I] index 変更対象を示す HDU インデックス
[I] name セットするバージョン番号
([I]：入力，[O]：出力)

**RETURN VALUE**

自身の参照を返します.

**EXAMPLES**

```
fits.assign_hduver(index, 101);
```

---

### 12.3.25 assign_hdulevel(), assign_extlevel()

**NAME**

assign_hdulevel() — HDU のレベル番号の変更.

**SYNOPSIS**

```
fitscc &assign_hdulevel( long index, long long level );
fitscc &assign_extlevel( long index, long long level );
```

**DESCRIPTION**

index で指定された HDU のレベル番号を変更します. level はヘッダ EXTLEVEL に反映されます.

Primary HDU の場合も指定可能です.

**PARAMETER**

[I] index 変更対象を示す HDU インデックス
[I] name セットするレベル番号
([I]：入力, [O]：出力)

**RETURN VALUE**

自身の参照を返します.

**EXAMPLES**

```
fits.assign_hdulevel(index, 1234);
```

---

### 12.3.26 hduver_is_set(), extver_is_set()

**NAME**

hduver_is_set() — HDU のバージョン番号が設定されているか判定します

**SYNOPSIS**

```
bool hduver_is_set( long index ) const;
bool hduver_is_set( const char *hduname ) const;
bool extver_is_set( long index ) const;
bool extver_is_set( const char *extname ) const;
```

**DESCRIPTION**

index で指定された HDU のバージョン番号が設定されているか判定します.

**PARAMETER**

[I] index HDU のインデックス
[I] hduname HDU 名
[I] extname HDU 名
([I]：入力, [O]：出力)

**RETURN VALUE**

HDU のバージョン番号が設定されていれば true，それ以外は false を返します.

**EXAMPLES**

次のコードは，全ての HDU に対してバージョン番号の設定を調べています．バージョン番号が設定されていなければバージョン 100 を設定します.

```
long i;
for ( i=0 ; i < fits.length() ; i++ ) {
    if ( fits.hduver_is_set(i) == false ) {
        fits.assign_hduver(i, 100);
    }
}
```

---

### 12.3.27 hdulevel_is_set(), extlevel_is_set()

**NAME**

hdulevel_is_set() — HDU のレベル番号が設定されているか判定します

**SYNOPSIS**

```
bool hdulevel_is_set( long index ) const;
bool hdulevel_is_set( const char *hduname ) const;
bool extlevel_is_set( long index ) const;
bool extlevel_is_set( const char *extname ) const;
```

**DESCRIPTION**

`index` で指定された HDU のレベル番号が設定されているか判定します.

**PARAMETER**

[I] `index` HDU のインデックス
[I] `hduname` HDU 名
[I] `extname` HDU 名
([I]：入力，[O]：出力)

**RETURN VALUE**

HDU のレベル番号が設定されていれば true，それ以外は false を返します.

**EXAMPLES**

次のコードは，全ての HDU に対してレベル番号の有無を調べ，設定されていなければレベル 1234 を設定します.

```
long i;
for ( i=0 ; i < fits.length() ; i++ ) {
    if ( fits.hdulevel_is_set(i) == false ) {
        fits.assign_hdulevel(i, 1234);
    }
}
```

---

## 12.4 ヘッダの操作

ここでは，ヘッダを操作するための API を解説します．API は 2 種類に大別されます．1 つめのケースは，

```
value = fits.hdu( ... ).header_function( ... );
```

の形，2 つめのケースは

```
value = fits.hdu( ... ).header( ... ).function( ... );
value = fits.hdu( ... ).headerf( ... ).function( ... );
```

の形です[37)]．

hdu( ... ) の括弧内の引数には，HDU 番号 (long index) あるいは HDU 名 (const char *hduname) を指定します．また，「hdu( ... )」の部分は，Image HDU の場合「image( ... )」，Ascii Table HDU か Binary Table HDU の場合「table( ... )」として使う事もできます．

header( ... ) の括弧内の引数には，ヘッダのインデックス (long index) あるいはヘッダのキーワード (const char *keyword) を指定します．headerf( ... ) の括弧内の引数には，ヘッダのキーワードを libc の printf() 関数と同様に指定します．

このマニュアルではこれ以降，「hdu( ... )」「header( ... )」「headerf( ... )」の括弧内の引数はすべて同じですので，その部分の引数に関する記述は省略します．

SFITSIO では，ヘッダのキーワードは最大 54 文字まで使用でき，文字列値の長さの制限はありません (§9.1を参照)．ヘッダレコードが 80 文字以内に収まらない場合でも，適切な方法でファイルに保存されます．

### 12.4.1 hdu().header_length()

**NAME**

hdu().header_length() — ヘッダのレコード数

**SYNOPSIS**

```
long hdu( ... ).header_length() const;
```

**RETURN VALUE**

ヘッダのレコード数を返します．

**EXAMPLES**

次のコードは，ヘッダのレコード数を表示します．

```
fits_image &primary = fits.image("Primary");
printf("Record Count = %ld\n", primary.header_length());
```

---

### 12.4.2 hdu().header_index()

**NAME**

hdu().header_index() — ヘッダのヘッダレコード番号

[37)] 次のように，クラスの構造を正確に表現したメンバ関数の組み合わせで，ヘッダ関連の API を利用する事もできます．1 つめの形は，value = fits.hdu(...).header().*function*(...);，2 つめの形は，value = fits.hdu(...).header().at(...).*function*(...); とも書けます．詳細はヘッダファイル (fits_hdu.h, fits_header.h, fits_header_record.h) をご覧ください．

**SYNOPSIS**

```
long hdu( ... ).header_index( const char *keyword ) const;
long hdu( ... ).header_index( const char *keyword, bool is_description ) const;
```

**DESCRIPTION**

記述形式 (COMMENT や HISTORY などの方法) ではないレコードからキーワード `keyword` を探してそのレコード番号を返します．`is_description` が `true` の場合は，記述形式のレコードから探します．

キーワードが見つからない場合は，負の数を返します．

**PARAMETER**

[I] `keyword` キーワード
[I] `is_description` 検索対象レコードの指定 (true:記述形式，false:それ以外)
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 ： レコード番号．
負の値 (エラー) ： 見つからなかった場合．

**EXAMPLES**

次のコードは，ヘッダのキーワード「TELESCOP」のインデックスを表示します．

```
fits_image &primary = fits.image("Primary");
printf("Record Index = %ld\n", primary.header_index("TELESCOP"));
```

---

### 12.4.3 hdu().header_regmatch()

**NAME**

hdu().header_regmatch() — ヘッダのキーワード検索

**SYNOPSIS**

```
long hdu( ... ).header_regmatch( long index, const char *keypat,
                                 ssize_t *rpos = NULL, size_t *rlen = NULL ) const;
long hdu( ... ).header_regmatch( const char *keypat,
                                 ssize_t *rpos = NULL, size_t *rlen = NULL ) const;
```

**DESCRIPTION**

`index` で示されたレコードから順に POSIX 拡張正規表現 `keypat` にマッチするキーワードを探し，そのレコード番号を返します．マッチするものが見つからない場合は，負の数を返します．`index` か無い場合は，最初のレコードから検索します．

`*rpos` には見つかった文字の位置を，`*rlen` にはマッチした文字列の長さを返します．これらの引数は与えなくてもかまいません．

このメンバ関数では，記述形式 (COMMENT や HISTORY などの方法) のヘッダレコードは検索対象になりません．

**PARAMETER**

[I] `index` 検索開始位置を示すヘッダレコードのインデックス
[I] `keypat` キーワードパターン文字列 (正規表現)
[O] `rpos` 検索結果の文字位置
[O] `rlen` 検索結果の文字列長
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 ： レコード番号.
負の値 (エラー) ： 見つからなかった場合.

**EXAMPLES**

次のコードは，プライマリ HDU のヘッダのキーワード名が `CRVAL1` あるいは `CRVAL2` から始まるレコードをすべて表示します.

```
fits_image &primary = fits.image("Primary");
long i = 0;
while ( 0 <= (i=primary.header_regmatch(i,"^CRVAL[1-2]")) ) {
    printf("%s = %s\n", primary.header(i).keyword(),
                        primary.header(i).value());
    i++;
}
```

チュートリアルの §5.6 にも使用例があります.

---

### 12.4.4 hdu().header().svalue()

**NAME**

hdu().header().svalue() — ヘッダの文字列の値 (高レベル)

**SYNOPSIS**

```
const char *hdu( ... ).header( ... ).svalue();
```

**DESCRIPTION**

指定されたヘッダレコードの文字列の値を，文字列をかこむシングルクォーテーション (') や余分な空白文字を取り除いてから返します.

返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，このメンバ関数が次に呼ばれた場合は無効になります.

**RETURN VALUE**

ヘッダ文字列のアドレスを返します.

**EXAMPLES**

次のコードは，プライマリ HDU のヘッダの名が `CTYPE1` のレコードの値を表示します.

```
fits_image &primary = fits.image("Primary");
printf("CTYPE1 = %s\n", primary.header("CTYPE1").svalue());
```

チュートリアルの §5.5 にも使用例があります.

---

### 12.4.5 hdu().header().get_svalue()

**NAME**

hdu().header().get_svalue() — ヘッダの文字列の値を得る (高レベル)

**SYNOPSIS**

```
size_t hdu( ... ).header( ... )
                 .get_svalue( char *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

指定されたヘッダレコードの文字列の値を，文字列をかこむシングルクォーテーション (') や余分な空白文字を取り除いてから，`dest_buf` へコピーします．`dest_buf` のバッファサイズは `buf_size` で指定します．

このメンバ関数は libc の `strncpy()` とは異なり，バッファのサイズが十分ではない場合でも，必ず`'\0'` で終端します．

**PARAMETER**

[O] `dest_buf` 文字列受取バッファのアドレス
[I] `buf_size` 文字列受取バッファのサイズ
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : バッファ長が十分な場合にコピーできる文字数 (`'\0'` は含まない).
負の値 (エラー) : 引数不正でコピーされなかった場合.

**EXAMPLES**

次のコードは，プライマリ HDU のヘッダの名が `CTYPE1` のレコードの値を用意したバッファに取得します．

```
char dest_buf[128];
fits_image &primary = fits.image("Primary");

primary.header("CTYPE1").get_svalue(dest_buf, sizeof(dest_buf));
```

---

### 12.4.6 hdu().header().dvalue()

**NAME**

hdu().header().dvalue() — ヘッダの実数値 (高レベル)

**SYNOPSIS**

```
double hdu( ... ).header( ... ).dvalue() const;
```

**RETURN VALUE**

指定されたヘッダレコードの実数値を返します．

**EXAMPLES**

次のコードは，プライマリ HDU のヘッダの `CDELT1` の値を取得します．

```
fits_image &primary = fits.image("Primary");
double value;

value = primary.header("CDELT1").dvalue();
```

チュートリアルの §5.5 にも使用例があります.

---

### 12.4.7 hdu().header().lvalue(), hdu().header().llvalue()

**NAME**

hdu().header().lvalue(), hdu().header().llvalue() — ヘッダの整数値 (高レベル)

**SYNOPSIS**

```
long hdu( ... ).header( ... ).lvalue() const;
long long hdu( ... ).header( ... ).llvalue() const;
```

**RETURN VALUE**

指定されたヘッダレコードの整数値を返します.

**EXAMPLES**

§12.4.6の EXAMPLES を参照してください.

チュートリアルの §5.5 にも使用例があります.

---

### 12.4.8 hdu().header().bvalue()

**NAME**

hdu().header().bvalue() — ヘッダの論理値 (高レベル)

**SYNOPSIS**

```
bool hdu( ... ).header( ... ).bvalue() const;
```

**RETURN VALUE**

指定されたヘッダレコードの論理値を返します.

**EXAMPLES**

§12.4.6の EXAMPLES を参照してください.

チュートリアルの §5.5 にも使用例があります.

---

### 12.4.9 hdu().header().assign(), hdu().header().assignf()

**NAME**

hdu().header().assign() — ヘッダに文字列の値を代入 (高レベル)

**SYNOPSIS**

```
fits_header_record &hdu( ... ).header( ... ).assign( const char *str );
fits_header_record &hdu( ... ).header( ... ).assignf( const char *format, ... );
```

**DESCRIPTION**

指定されたヘッダレコードに，文字列の値 `str` をセットします.

新規にヘッダレコードを追加する場合は，`header()` の引数にキーワードをセットしてください.

文字列値の長さの制限はありません (§9.1を参照)．ヘッダレコードが 80 文字以内に収まらない場合でも，適切な方法でファイルに保存されます.

`format` 以降の引数は，libc の `printf()` のそれと同様に指定します.

`format` 中の `%` に続く変換文字とその機能は次の表のとおりです．なお，`%` 自体を出力したい場合は `%%` とします.

| 変換文字 | 変換の内容 | 引数の型 |
|---|---|---|
| hhd | 引数を符号付き 10 進数に変換する | char 型 |
| hd | 引数を符号付き 10 進数に変換する | short 型 |
| d | 引数を符号付き 10 進数に変換する | int 型 |
| ld | 引数を符号付き 10 進数に変換する | long 型 |
| lld | 引数を符号付き 10 進数に変換する | long long 型 |
| zd | 引数を符号付き 10 進数に変換する | ssize_t 型 |
| hhu | 引数を符号なし 10 進数に変換する | unsigned char 型 |
| hu | 引数を符号なし 10 進数に変換する | unsigned short 型 |
| u | 引数を符号なし 10 進数に変換する | unsigned int 型 |
| lu | 引数を符号なし 10 進数に変換する | unsigned long 型 |
| llu | 引数を符号なし 10 進数に変換する | unsigned long long 型 |
| zu | 引数を符号なし 10 進数に変換する | size_t 型 |
| hho | 引数を符号なし 8 進数に変換する | (unsigned) char 型 |
| ho | 引数を符号なし 8 進数に変換する | (unsigned) short 型 |
| o | 引数を符号なし 8 進数に変換する | (unsigned) int 型 |
| lo | 引数を符号なし 8 進数に変換する | (unsigned) long 型 |
| llo | 引数を符号なし 8 進数に変換する | (unsigned) long long 型 |
| zo | 引数を符号なし 8 進数に変換する | size_t 型, ssize_t 型 |
| hhx, hhX | 引数を符号なし 16 進数に変換する | (unsigned) char 型 |
| hx, hX | 引数を符号なし 16 進数に変換する | (unsigned) short 型 |
| x, X | 引数を符号なし 16 進数に変換する | (unsigned) int 型 |
| lx, lX | 引数を符号なし 16 進数に変換する | (unsigned) long 型 |
| llx, llX | 引数を符号なし 16 進数に変換する | (unsigned) long long 型 |
| zx, zX | 引数を符号なし 16 進数に変換する | size_t 型, ssize_t 型 |
| c | 引数を 1 文字として出力する | int 型 |
| s | 引数によって指し示される文字列を出力する．出力されるのは null 文字の前の文字までか，指示された最大の文字数分 | const char *型 |
| f | 引数を float か double として受けとり [-] ○○○. ○○○の形の 10 進数に変換する | 浮動小数点型 |
| e, E | 引数を float か double として受けとり [-] ○. ○○○○○ e[±] ○○の形の 10 進数に変換する | 浮動小数点型 |
| g, G | %e あるいは %f による変換の文字数の少ない方の変換をとる | 浮動小数点型 |
| a, A | 引数を float か double として受けとり [-]0x ○. ○○○○○ p[±] ○○の形の 16 進数に変換する | 浮動小数点型 |
| p | 引数を void*ポインタとして受けとり 16 進数で出力する | void*型 |
| n | これまでに出力された文字数を int*ポインタ引数が指す整数に保存する | int*型 |

さらに，% と変換文字との間に，次のオプション指示子を入れると，より詳細な書式指定ができます．

```
sio.printf("%__ f…\n",x);
```

↑

$\overbrace{[-][+][[0]m][.][n]}$

| オプション | 意味 | 例 |
|---|---|---|
| − (マイナス符号) | 変換された引数を左詰めで出力する． | `.printf("%-6d…` |
| | 1234_____ | |
| + | 符号付き変換された数値の前に，常に符号 (+ か -) をつける | `.printf("%+6d…` |
| $m$ (桁数記入) | 確保すべき領域幅を最小でも $m$ 字の広さにとる． | `.printf("%10d…` |
| | 領域幅に空白が生じる時は，左側にブランクが入る．0 をつけるとゼロパディングする． | `.printf("%010d…` |
| | _____1234 | |
| . (ピリオド) | フィールド幅と文字数または小数点以下の桁数との区切り | `.printf("%10.5f…` |
| $n$ (桁数記入) | f 指定の場合は，小数点以下の桁数． | `.printf("%10.5f…` |
| | e，E，g，G 指定の場合は，精度． | `.printf("%.15g…` |
| | 文字列の場合はその取り幅 (文字の頭から) | `.printf("%10.5s…` |

**PARAMETER**

[I] `str` セットする文字列
[I] `format` フォーマット文字列
[I] `...` フォーマットの全要素データ
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_header_record への参照を返します．

**EXCEPTION**

内部バッファの確保に失敗した場合や，各要素データが指定された変換フォーマットで変換できない値の場合，SLLIB に由来する例外 (`sli::err_rec` 例外) が発生します．

**EXAMPLES**

次のコードは，プライマリ HDU にキーワード `TELESCOP`，値 `HST` のヘッダレコードを登録します．新規の場合だけでなく，値の更新の場合もこの書き方が使えます．

```
fits.image("Primary").header("TELESCOP").assign("HST");
```

チュートリアルの §5.5 にも使用例があります．

---

### 12.4.10 hdu().header().assign()

**NAME**

hdu().header().assign() — ヘッダに論理値を代入 (高レベル)

**SYNOPSIS**

```
fits_header_record &hdu( ... ).header( ... ).assign( bool value );
```

**DESCRIPTION**

指定されたヘッダレコードに，論理値 value をセットします.

新規にヘッダレコードを追加する場合は，header() の引数にキーワードをセットしてください.

**PARAMETER**

[I] value セットする論理値

([I]：入力，[O]：出力)

**EXCEPTION**

内部バッファの確保や操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**RETURN VALUE**

当該 fits_header_record への参照を返します.

**EXAMPLES**

§12.4.9の EXAMPLES を参照してください.

チュートリアルの §5.5 にも使用例があります.

---

### 12.4.11 hdu().header().assign()

**NAME**

hdu().header().assign() — ヘッダに整数値を代入 (高レベル)

**SYNOPSIS**

```
fits_header_record &hdu( ... ).header( ... ).assign( int value );
fits_header_record &hdu( ... ).header( ... ).assign( long value );
fits_header_record &hdu( ... ).header( ... ).assign( long long value );
```

**DESCRIPTION**

指定されたヘッダレコードに，整数値 value をセットします.

新規にヘッダレコードを追加する場合は，header() の引数にキーワードをセットしてください.

**PARAMETER**

[I] value セットする整数値

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_header_record への参照を返します.

**EXCEPTION**

内部バッファの確保や操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，プライマリ HDU にキーワード CCDPICNO，値 35 のヘッダレコードを登録します．新規の場合だけでなく，値の更新の場合もこの書き方が使えます．

```
fits.image("Primary").header("CCDPICNO").assign(35);
```

チュートリアルの §5.5 にも使用例があります．

---

### 12.4.12 hdu().header().assign()

**NAME**

hdu().header().assign() — ヘッダに実数値を代入 (高レベル)

**SYNOPSIS**

```
fits_header_record &hdu( ... ).header( ... ).assign( double value, int prec = 15 );
fits_header_record &hdu( ... ).header( ... ).assign( float value, int prec = 6 );
```

**DESCRIPTION**

指定されたヘッダレコードに，実数値 value をセットします．prec には桁数を指定できます．省略した場合，value が double 型の場合は 15 桁の精度で，value が float 型の場合は 6 桁の精度で，ヘッダレコードに書き込みます．

新規にヘッダレコードを追加する場合は，header() の引数にキーワードをセットしてください．

**PARAMETER**

[I] value セットする実数値
[I] prec 精度 (桁数)
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_header_record への参照を返します．

**EXCEPTION**

内部バッファの確保や操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

次のコードは，プライマリ HDU にキーワード CDELT1，値-0.01 のヘッダレコードを登録します．新規の場合だけでなく，値の更新の場合もこの書き方が使えます．

```
fits.image("Primary").header("CDELT1").assign(-0.01);
```

チュートリアルの §5.5 にも使用例があります．

---

### 12.4.13 hdu().header().type()

**NAME**

hdu().header().type() — ヘッダのレコードの型

**SYNOPSIS**

```
int hdu( ... ).header( ... ).type() const;
```

**DESCRIPTION**

指定されたヘッダレコードの型を調べます.

実数ならFITS::DOUBLE_Tが, 整数ならFITS::LONGLONG_Tが, 複素数ならFITS::DOUBLECOMPLEX_T
論理値ならFITS::BOOL_Tが, それ以外ならFITS::STRING_Tが返ります.

**RETURN VALUE**

| | | |
|---|---|---|
| FITS::DOUBLE_T | : | ヘッダレコードが実数の場合. |
| FITS::LONGLONG_T | : | ヘッダレコードが整数の場合. |
| FITS::DOUBLECOMPLEX_T | : | ヘッダレコードが複素数の場合. |
| FITS::BOOL_T | : | ヘッダレコードが論理数の場合. |
| FITS::STRING_T | : | それ以外の場合. |

**EXAMPLES**

次のコードはヘッダのレコード型を調べています.

```
switch ( fits.hdu("Primary").header("TELESCOP").type() ) {
  case FITS::DOUBLE_T:
      printf("レコードは実数型です\n");
      break;
  case FITS::LONGLONG_T:
      printf("レコードは整数型です\n");
      break;
  case FITS::DOUBLECOMPLEX_T:
      printf("レコードは複素数型です\n");
      break;
  case FITS::BOOL_T:
      printf("レコードは論理型です\n");
      break;
  case FITS::STRING_T:
      printf("レコードは文字列型です\n");
      break;
  default:
      printf("不明なレコード型です\n");
      break;
}
```

### 12.4.14 hdu().header().status()

**NAME**

hdu().header().status() — ヘッダのレコードの状態

**SYNOPSIS**

```
bool hdu( ... ).header( long index ).status() const;
```

**DESCRIPTION**

index で指定されたヘッダレコードが，通常の形式 (A = B の形式) か，記述形式 (COMMENT や HISTORY などの記述) か，NULL 形式 (キーワードも値も無い状態) かどうかを調べ，それぞれ FITS::NORMAL_RECORD，FITS::DESCRIPTION_RECORD，FITS::NULL_RECORD を返します．

**RETURN VALUE**

| | | |
|---|---|---|
| FITS::NORMAL_RECORD | : | ヘッダレコードが通常の形式の場合． |
| FITS::DESCRIPTION_RECORD | : | ヘッダレコードが記述形式の場合． |
| FITS::NULL_RECORD | : | ヘッダレコードが NULL 形式の場合． |

**EXAMPLES**

次のコードはヘッダレコードの状態を調べています．

```
switch ( fits.hdu("Primary").header("TELESCOP").status() ) {
  case FITS::NORMAL_RECORD:
      printf("レコードは通常の形式です\n");
      break;
  case FITS::DESCRIPTION_RECORD:
      printf("レコードは記述形式です\n");
      break;
  case FITS::NULL_RECORD:
      printf("レコードは NULL 形式です\n");
      break;
  default:
      printf("これは表示されないはず\n");
      break;
}
```

---

### 12.4.15 hdu().header().keyword()

**NAME**

hdu().header().keyword() — ヘッダのキーワード名

**SYNOPSIS**

```
const char *hdu( ... ).header( long index ).keyword() const;
```

**DESCRIPTION**

index で指定されたヘッダレコードのキーワード名を返します．

返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，ヘッダレコードが変更された場合は無効になります．

**RETURN VALUE**

ヘッダレコードのキーワード名文字列のアドレスを返します.

**EXAMPLES**

次のコードは，プライマリ HDU の index で指定されたヘッダのキーワードと値のセットを標準出力に表示します.

```
fits_image &primary = fits.image("Primary");
printf("%s = %s\n",
       primary.header(index).keyword(), primary.header(index).value());
```

---

### 12.4.16 hdu().header().get_keyword()

**NAME**

hdu().header().get_keyword() — ヘッダのキーワード名を文字列の値を得る (高レベル)

**SYNOPSIS**

```
ssize_t hdu( ... ).header( long index )
                  .get_keyword( char *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

index で指定されたヘッダレコードのキーワード名を dest_buf へコピーします. dest_buf のバッファサイズは buf_size で指定します.

このメンバ関数は libc の strncpy() とは異なり，バッファのサイズが十分ではない場合でも，必ず'\0' で終端します.

**PARAMETER**

[O] dest_buf 文字列受取バッファのアドレス
[I] buf_size 文字列受取バッファのサイズ
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : バッファ長が十分な場合にコピーできる文字数 ('\0' は含まない).
負の値 (エラー) : 引数不正でコピーされなかった場合.

**EXAMPLES**

次のコードは，プライマリ HDU の index で指定されたヘッダのキーワードを用意したバッファに取得します.

```
char dest_buf[128];
fits_image &primary = fits.image("Primary");
primary.header(index).get_keyword(dest_buf, sizeof(dest_buf));
```

---

### 12.4.17 hdu().header().value()

**NAME**

hdu().header().value() — ヘッダの値 (低レベル)

**SYNOPSIS**

```
const char *hdu( ... ).header( ... ).value() const;
```

**DESCRIPTION**

指定されたヘッダレコードの生の値を返します.

文字列をかこむシングルクォーテーション (') や余分な空白文字を取り除いた値が必要な場合は, `hdu().header().svalue()` (§12.4.4) を使います.

返される値はオブジェクト内部バッファのアドレスですから, オブジェクトが破棄されたり, ヘッダレコードが変更された場合は無効になります.

**RETURN VALUE**

ヘッダレコード値文字列のアドレスを返します.

**EXAMPLES**

§12.4.15の EXAMPLES を参照してください.

### 12.4.18 hdu().header().get_value()

**NAME**

hdu().header().get_value() — ヘッダの値 (低レベル)

**SYNOPSIS**

```
ssize_t hdu( ... ).header( ... )
                  .get_value( char *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

指定されたヘッダレコードの生の値を `dest_buf` へコピーします. `dest_buf` のバッファサイズは `buf_size` で指定します.

文字列をかこむシングルクォーテーション (') や余分な空白文字を取り除いた値が必要な場合は, `hdu().header().get_svalue()` (§12.4.5) を使います.

このメンバ関数は libc の `strncpy()` とは異なり, バッファのサイズが十分ではない場合でも, 必ず`'\0'` で終端します.

**PARAMETER**

[O] `dest_buf` 文字列受取バッファのアドレス
[I] `buf_size` 文字列受取バッファのサイズ
([I]：入力, [O]：出力)

**RETURN VALUE**

非負の値 : バッファ長が十分な場合にコピーできる文字数 (`'\0'` は含まない).
負の値 (エラー) : 引数不正でコピーされなかった場合.

**EXAMPLES**

§12.4.16の EXAMPLES を参照してください.

### 12.4.19 hdu().header().comment()

**NAME**

hdu().header().comment() — ヘッダのコメント

**SYNOPSIS**

```
const char *hdu( ... ).header( ... ).comment() const;
```

**DESCRIPTION**

指定されたヘッダレコードのコメント文字列を返します.

返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，ヘッダレコードが変更された場合は無効になります.

**RETURN VALUE**

コメント文字列のアドレスを返します.

**EXAMPLES**

§12.4.15の EXAMPLES を参照してください.

---

### 12.4.20 hdu().header().get_comment()

**NAME**

hdu().header().get_comment() — ヘッダのコメントを取得

**SYNOPSIS**

```
ssize_t hdu( ... ).header( ... )
                  .get_comment( char *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

指定されたヘッダレコードのコメント文を `dest_buf` へコピーします. `dest_buf` のバッファサイズは `buf_size` で指定します.

このメンバ関数は libc の `strncpy()` とは異なり，バッファのサイズが十分ではない場合でも，必ず`'\0'` で終端します.

**PARAMETER**

[O] `dest_buf` 文字列受取バッファのアドレス
[I] `buf_size` 文字列受取バッファのサイズ
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : バッファ長が十分な場合にコピーできる文字数 (`'\0'` は含まない).
負の値 (エラー) : 引数不正でコピーされなかった場合.

**EXAMPLES**

§12.4.16の EXAMPLES を参照してください.

---

### 12.4.21 hdu().header().assign_value()

**NAME**

hdu().header().assign_value() — ヘッダに文字列を代入 (低レベル)

**SYNOPSIS**

```
fits_header_record &hdu( ... ).header( ... ).assign_value( const char *value );
fits_header_record &hdu( ... ).header( ... )
                               .assignf_value( const char *format, ... );
```

**DESCRIPTION**

指定されたヘッダレコードの値に生の値 value をセットします．ヘッダのレコードタイプを文字列としたい場合は，value に"'ABC'"のようにセットします．レコードタイプを数字や論理値にする場合は，value に"256"，"3.14"，"T" のようにセットします．

新規にヘッダレコードを追加する場合は，header() の引数にキーワードをセットしてください．

文字列値の長さの制限はありません (§9.1を参照)．ヘッダレコードが 80 文字以内に収まらない場合でも，適切な方法でファイルに保存されます．

format 以降の引数は，libc の printf() のそれと同様に指定します．

**PARAMETER**

[I] value セットする文字列
[I] format フォーマット文字列
[I] ... フォーマットの全要素データ
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_header_record への参照を返します．

**EXCEPTION**

内部バッファの確保に失敗した場合や，各要素データが指定された変換フォーマットで変換できない値の場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
fits.hdu("Primary").header("TELESCOP").assign_value("'HST'");
```

---

### 12.4.22 hdu().header().assign_comment()

**NAME**

hdu().header().assign_comment() — ヘッダのコメントを変更

**SYNOPSIS**

```
fits_header_record &hdu( ... ).header( ... ).assign_comment( const char *comment );
fits_header_record &hdu( ... ).header( ... )
                               .assignf_comment( const char *format, ... );
```

**DESCRIPTION**

指定されたヘッダレコードのコメントに，文字列 comment をセットします．

新規にヘッダレコードを追加する場合は，header() の引数にキーワードをセットしてください．

format 以降の引数は，libc の printf() のそれと同様に指定します．

**PARAMETER**

[I] comment セットするコメント文字列
[I] format フォーマット文字列
[I] ... フォーマットの全要素データ
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_header_record への参照を返します．

**EXCEPTION**

内部バッファの確保に失敗した場合や，各要素データが指定された変換フォーマットで変換できない値の場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
fits.hdu("Primary").header("TELESCOP").assign("HST")
                                      .assign_comment("Name of telescope");
```

チュートリアルの §5.5 にも使用例があります．

---

### 12.4.23 hdu().header_update()

**NAME**

hdu().header_update() — 1 つのヘッダレコードを更新または追加

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_update( const char *keyword,
                                    const char *value, const char *comment );
```

**DESCRIPTION**

keyword で指定されたレコードを，生の値を value で，コメントを comment で更新します．更新が不要な場合，value か comment のどちらかに NULL を与えます．keyword がヘッダに見つからない場合は追加します．

ヘッダのレコードタイプを文字列としたい場合は，value に"'ABC'"のようにセットします．レコードタイプを数字や論理値にする場合は，value に"256"，"3.14"，"T" のようにセットします．

**PARAMETER**

[I] keyword キーワード
[I] value セットする文字列値
[I] comment セットするコメント値
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの確保に失敗した場合 (例えばメモリ不足で追加レコード分の領域確保に失敗した場合)，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.hdu("Primary").header_update("TELESCOP", "HST", "Name of telescope");
```

---

### 12.4.24 hdu().header_assign()

**NAME**

hdu().header_assign() — ヘッダレコードを更新

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_assign( long index, const fits::header_def &def );
fits_hdu &hdu( ... ).header_assign( const char *header_keyword,
                                    const fits::header_def &def );
fits_hdu &hdu( ... ).header_assign( long index, const fits_header_record &obj );
fits_hdu &hdu( ... ).header_assign( const char *header_keyword,
                                    const fits_header_record &obj );
fits_hdu &hdu( ... ).header_assign( long index,
                  const char *keyword, const char *value, const char *comment );
fits_hdu &hdu( ... ).header_assign( const char *header_keyword,
                  const char *keyword, const char *value, const char *comment );
fits_hdu &hdu( ... ).header_assign( long index,
                                  const char *keyword, const char *description );
fits_hdu &hdu( ... ).header_assign( const char *header_keyword,
                                  const char *keyword, const char *description );
```

**DESCRIPTION**

`index` または `header_keyword` で指定されたヘッダレコードを各引数で指定された内容に更新します.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `index` | 更新対象を示すヘッダレコードのインデックス |
| [I] | `header_keyword` | 更新対象を示すヘッダレコードのキーワード |
| [I] | `def` | コピー元となる`fits::table_def` 構造体 |
| [I] | `obj` | コピー元となるオブジェクト |
| [I] | `keyword` | セットするキーワード値 |
| [I] | `value` | セットするレコード値 |
| [I] | `comment` | セットするコメント値 |
| [I] | `description` | セットする Description 値 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，index で指定されたヘッダレコードを更新します.

```
fits.hdu("Primary").header_assign(index, "TELESCOP", "HST", "Name of telescope");
```

---

### 12.4.25 hdu().header_init()

**NAME**

hdu().header_init() — ヘッダの初期化

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_init();
fits_hdu &hdu( ... ).header_init( const fits::header_def defs[] );
fits_hdu &hdu( ... ).header_init( const fits_header &obj );
```

**DESCRIPTION**

ヘッダを消去し，defs が指定された場合はその内容をヘッダにセットします.

fits::header_def の定義は次の通りです.

```
typedef struct {
    const char *keyword;
    const char *value;
    const char *comment;
} fits::header_def;
```

defs の配列の最後のメンバはすべて NULL にします.

**PARAMETER**

[I] defs コピー元となるfits::table_def 構造体の配列

[I] obj コピー元となるオブジェクト

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えばメモリ不足で指定されたレコード分の領域を確保出来なかった場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits::header_def defs[] = { {"TELESCOP", "'SUBARU'", "Name of telescope"},
                            {"OBSERVAT", "'NAOJ'", "Observatory name"},
                            {"COMMENT", "--------------------------------"},
                            {NULL} };
fits.hdu("Primary").header_init(defs);
```

値が文字列の場合，"'SUBARU'" あるいは "SUBARU" のどちらでもかまいません．ただし，123 のような数字を文字列としてファイルに格納したい場合は "'123'" のように値をセットします．

コメントやヒストリの記述の場合は，value か comment のどちらかを NULL にしておきます．

チュートリアルの §5.9 にも使用例があります．

---

### 12.4.26 hdu().header_swap()

**NAME**

hdu().header_swap() — ヘッダのスワップ

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_swap( fits_header &obj );
```

**DESCRIPTION**

fits_header オブジェクト obj と内容を入れ替えます．Data Unit のための予約キーワードについては，入れ替えは行なわれません．

**PARAMETER**

[I/O] obj 入れ替え対象のオブジェクト

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します．

**EXAMPLES**

次のコードは HDU「AAAA」と HDU「BBBB」のヘッダを入れ替えます．

```
fits_header &obj = fits.hdu("BBBB").header();
fits.hdu("AAAA").header_swap(obj);
```

---

### 12.4.27 hdu().header_append_records()

**NAME**

hdu().header_append_records() — ヘッダレコードを追加

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_append_records( const fits::header_def defs[] );
fits_hdu &hdu( ... ).header_append_records( const fits_header &obj );
```

**DESCRIPTION**

hdu().header_append_records() メンバ関数は，defs の内容をヘッダに追加します．defs の配列の最後のメンバはすべて NULL にします．fits::header_def の仕様については，hdu().header_init() に関する解説を参照してください．

**PARAMETER**

[I] defs 追加時にコピー元となるfits::table_def 構造体の配列

[I] obj 追加時にコピー元となるヘッダオブジェクト

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えばメモリ不足で追加レコード分の領域を確保出来なかった場合), SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits::header_def defs[] = { {"TELESCOP", "'SUBARU'", "Name of telescope"},
                            {"OBSERVAT", "'NAOJ'", "Observatory name"},
                            {"COMMENT", "--------------------------------"},
                            {NULL} };
fits.hdu("Primary").header_append_records(defs);
```

チュートリアルの §5.7 にも使用例があります.

---

### 12.4.28 hdu().header_append()

**NAME**

hdu().header_append() — ヘッダレコードを追加

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_append( const char *keyword );
fits_hdu &hdu( ... ).header_append( const char *keyword, const char *value,
                                    const char *comment );
fits_hdu &hdu( ... ).header_append( const char *keyword, const char *description );
fits_hdu &hdu( ... ).header_append( const fits::header_def &def );
fits_hdu &hdu( ... ).header_append( const fits_header_record &obj );
```

**DESCRIPTION**

hdu().header_append() メンバ関数は, キーワード `keyword`, 値 `value`, コメント `comment` のヘッダレコードを追加します. `description` が指定された場合は, 記述形式 (COMMENT や HISTORY などの仕様) で追加します.

`value` と `description` に文字列長の制限はありません (§9.1を参照). ヘッダレコードが 80 文字以内に収まらない場合でも, それぞれ適切な方法でファイルに保存されます.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `keyword` | 追加されるレコードのキーワード値 |
| [I] | `value` | 追加されるレコードの値 |
| [I] | `comment` | 追加されるレコードのコメント |
| [I] | `description` | 追加されるレコードの Description 値 |
| [I] | `def` | 追加される`fits::table_def` 構造体 |
| [I] | `obj` | 追加時にコピー元となるヘッダレコードオブジェクト |

([I]：入力, [O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えばメモリ不足で追加レコード分の領域を確保出来なかった場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits.hdu("Primary").header_append("TELESCOP", "'SUBARU'", "Name of telescope");
```

チュートリアルの §5.7 にも使用例があります.

---

### 12.4.29 hdu().header_insert_records()

**NAME**

hdu().header_insert_records() — ヘッダレコードを挿入

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_insert_records( long index,
                                            const fits::header_def defs[] );
fits_hdu &hdu( ... ).header_insert_records( const char *keyword,
                                            const fits::header_def defs[] );
fits_hdu &hdu( ... ).header_insert_records( long index, const fits_header &obj );
fits_hdu &hdu( ... ).header_insert_records( const char *keyword,
                                            const fits_header &obj );
```

**DESCRIPTION**

hdu().header_insert_records() メンバ関数は，defs の内容を，index あるいは keyword で指定されたレコードに挿入します. defs の配列の最後のメンバはすべて NULL にします. fits::header_def の仕様については，hdu().header_init() に関する解説を参照してください.

**PARAMETER**

[I] index 挿入位置を示すレコードのインデックス
[I] keyword 挿入位置を示すレコードのキーワード
[I] defs コピー元となるfits::table_def 構造体の配列
[I] obj コピー元となるオブジェクト
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えばメモリ不足で追加レコード分の領域を確保出来なかった場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits::header_def defs[] = { {"TELESCOP", "'SUBARU'", "Name of telescope"},
                            {"OBSERVAT", "'NAOJ'", "Observatory name"},
                            {"COMMENT", "--------------------------------"},
                            {NULL} };
long insert_index = 1;
fits.hdu("Primary").header_insert_records(insert_index, defs);
```

### 12.4.30 hdu().header_insert()

**NAME**

hdu().header_insert() — ヘッダレコードを挿入

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_insert( long index,
                                    const char *keyword,
                                    const char *value, const char *comment );
fits_hdu &hdu( ... ).header_insert( const char *record_keyword,
                                    const char *keyword,
                                    const char *value, const char *comment );
fits_hdu &hdu( ... ).header_insert( long index,
                                    const char *keyword, const char *description );
fits_hdu &hdu( ... ).header_insert( const char *record_keyword,
                                    const char *keyword, const char *description );
fits_hdu &hdu( ... ).header_insert( long index, const fits::header_def &def );
fits_hdu &hdu( ... ).header_insert( const char *record_keyword,
                                    const fits::header_def &def );
fits_hdu &hdu( ... ).header_insert( long index, const fits_header_record &obj );
fits_hdu &hdu( ... ).header_insert( const char *record_keyword,
                                    const fits_header_record &obj );
```

**DESCRIPTION**

hdu().header_insert() メンバ関数は，index0 または keyword0 で指定された位置に，キーワード keyword，値 value，コメント comment のヘッダレコードを挿入します．description が指定された場合は，記述形式 (COMMENT や HISTORY などの方法) で挿入します．

**PARAMETER**

| | | |
|---|---|---|
| [I] | index | 挿入位置を示すヘッダレコードのインデックス |
| [I] | record_keyword | 挿入位置を示すヘッダレコードのキーワード |
| [I] | def | コピー元となるfits::header_def 構造体 |
| [I] | obj | コピー元となるオブジェクト |
| [I] | keyword | セットするキーワード |
| [I] | value | セットするレコード値 |
| [I] | comment | セットするコメント値 |
| [I] | description | セットする Description 値 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えばメモリ不足で追加レコード分の領域を確保出来なかった場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
long insert_index = 1;
fits.hdu("Primary").header_insert(insert_index, "TELESCOP",
                                  "'SUBARU'", "Name of telescope");
```

---

### 12.4.31 hdu().header_erase_records()

**NAME**

hdu().header_erase_records() — ヘッダレコードを削除

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_erase_records( long index, long num_records );
fits_hdu &hdu( ... ).header_erase_records( const char *keyword, long num_records );
```

**DESCRIPTION**

hdu().header_erase_records() メンバ関数は，index あるいは keyword で指定されたレコードから，num_records 個のヘッダレコードを削除します.

**PARAMETER**

[I] index 削除開始位置を示すヘッダレコードのインデックス
[I] keyword 削除開始位置を示すヘッダレコードのキーワード
[I] num_records 削除レコード数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば削除後の領域リサイズ処理で失敗した場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，PrimaryHDU のヘッダのインデックス 0 番目のレコードから 2 つのレコードを削除します.

```
fits.hdu("Primary").header_erase_records(0L, 2);
```

---

### 12.4.32 hdu().header_erase()

**NAME**

hdu().header_erase() — ヘッダレコードを削除

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_erase( long index );
fits_hdu &hdu( ... ).header_erase( const char *keyword );
```

**DESCRIPTION**

hdu().header_erase() メンバ関数は，index または keyword で指定されたヘッダレコードを削除します.

**PARAMETER**

[I] index 削除対象を示すヘッダレコードのインデックス
[I] keyword 削除対象を示すヘッダレコードのキーワード
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば削除後の領域リサイズ処理で失敗した場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，PrimaryHDU のヘッダのインデックス 0 番目のレコードを削除します.

```
fits.hdu("Primary").header_erase(0L);
```

---

### 12.4.33 hdu().header_rename()

**NAME**

hdu().header_rename() — ヘッダレコードのキーワードを変更

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_rename( long index0, const char *new_name );
fits_hdu &hdu( ... ).header_rename( const char *keyword0, const char *new_name );
```

**DESCRIPTION**

hdu().header_rename() メンバ関数は，index0 または keyword0 で指定されたヘッダレコードのキーワードを new_name で指定されたものに変更します.

**PARAMETER**

[I] index0 変更対象を示すヘッダレコードのインデックス
[I] keyword0 変更対象を示すヘッダレコードのキーワード
[I] new_name 変更後のキーワード
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，PrimaryHDU のヘッダのキーワード “RADECSYS” を “RADESYS” に変更します.

```
fits.hdu("Primary").header_rename("RADECSYS", "RADESYS");
```

---

### 12.4.34 hdu().header()

**NAME**

hdu().header() — ヘッダオブジェクトの参照

**SYNOPSIS**

```
fits_header &hdu( ... ).header();
```

**RETURN VALUE**

ヘッダオブジェクトへの参照を返します.

**EXAMPLES**

次のコードは，Primary HDU のヘッダオブジェクトの参照をコピーしています.

```
fits_header &primary_header = fits.hdu("Primary").header();
```

---

### 12.4.35 hdu().header_formatted_string()

**NAME**

hdu().header_formatted_string() — ヘッダのフォーマット済み文字列

**SYNOPSIS**

```
const char *hdu( ... ).header_formatted_string();
```

**DESCRIPTION**

ヘッダの全レコードについて，FITS ファイル出力時のフォーマット済み文字列をオブジェクト内部に作成し，そのアドレスを返します．返される文字列は、改行を含まない'\0' で終端される 80 × $n$ キャラクタの文字列であり、ロング値は CONTINUE キーワードによって分割された形に整形されます。

WCSLIB と連携する時に使うと便利です.

**RETURN VALUE**

フォーマット済み文字列のアドレスを返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，プライマリ HDU のヘッダの全レコードのフォーマット済み文字列の行数と文字列すべてを表示します．tstring クラスを使うと，文字列操作が簡単に行なえます.

```
tstring hdr_all = fits.image("Primary").header_formatted_string();
printf("行数 = %d\n", (int)hdr_all.length() / 80);
printf("フォーマット済みヘッダ :\n");
printf("%s\n", hdr_all.cstr());
```

WCSLIB と連携については，チュートリアルの §5.13 にも使用例があります.

tstring クラスについては，APPENDIX3(§15) で解説しています.

---

### 12.4.36 hdu().header().get_section_info()

**NAME**

hdu().header().get_section_info() — IRAF 形式の矩形領域情報の取得

**SYNOPSIS**

```
int hdu( ... ).header( ... ).get_section_info( long dest_buf[], int buf_len ) const;
```

**DESCRIPTION**

FITS ヘッダにおける，IRAF 形式の矩形領域情報 (BIASSEC 等) をパースし，引数 dest_buf の配列に取得します.

例えば，次のようなヘッダレコード

BIASSEC = '[3074:3104,1:512] '

の場合，配列 dest_buf には (3073,31,0,512) のように，$x$ の位置 (0-indexed)，長さ，$y$ の位置 (0-indexed)，長さの順に値が返されます．これらの値は，ピクセルの統計を計算するための image().stat_pixels() メンバ関数 (§12.6.41) にそのまま与える事ができます.

**PARAMETER**

[O] dest_buf 結果を格納するための配列

[I] buf_len dest_buf のバッファ長

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : 取り出した値の個数 (通常は 4).

負の値 (エラー) : ヘッダレコードのパースに失敗した場合.

**EXAMPLES**

§1.1の課題 3 の解答例を参照してください.

---

### 12.4.37 hdu().header().assign_system_time()

**NAME**

hdu().header().assign_system_time() — 現在の日付時刻をセット

**SYNOPSIS**

```
fits_header_record &hdu( ... ).header( ... ).assign_system_time();
```

**DESCRIPTION**

現在の日付時刻 (UTC) を「yyyy-mm-ddThh:mm:ss」の形式でセットします.

**RETURN VALUE**

ヘッダオブジェクトへの参照を返します.

**EXAMPLES**

次のコードはヘッダに DATE キーワードを追加し，現在の日付時刻 (UTC) をその値としてセットします.

```
fits.image("Primary").header_append("DATE")
                     .header("DATE").assign_system_time();
```

---

### 12.4.38 hdu().header_fill_blank_comments()

**NAME**

hdu().header_fill_blank_comments() — コメント辞書に用意されたコメント文をセット

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_fill_blank_comments(int hdutype = FITS::ANY_HDU);
```

**DESCRIPTION**

全ヘッダレコード (ただし記述型レコードを除く) からコメントがセットされていないものを調べ，SFITSIO が持つコメント辞書に該当するキーワードがあればその内容でコメント文をセットします．このコメント辞書は，各 HDU タイプ専用の辞書が 3 つ，汎用の辞書が 1 つから構成され，どれを使うかは引数 hdutype で指定します．なお，それぞれの HDU 専用の辞書にキーワードが存在しない場合は，汎用のコメント辞書が使われます.

引数として指定できるのは，FITS::IMAGE_HDU, FITS::BINARY_TABLE_HDU, FITS::ASCII_TABLE_HDU, FITS::ANY_HDU，のいずれかで，FITS::ANY_HDU が指定された場合と引数を省略した場合は，HDU のタイプをヘッダの内容から自動判定します.

コメント辞書の変更については，§12.4.40 をご覧ください.

**PARAMETER**

[I] hdutype HDU のタイプ

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，プライマリ HDU のすべての空白コメントをコメント辞書の内容でセットします.

```
fits.image("Primary").header_fill_blank_comments(FITS::IMAGE_HDU);
```

---

### 12.4.39 hdu().header_assign_default_comments()

**NAME**

hdu().header_assign_default_comments() — コメント辞書に用意されたコメント文をセット (全て上書き)

**SYNOPSIS**

```
fits_hdu &hdu( ... ).header_assign_default_comments(int hdutype = FITS::ANY_HDU);
```

**DESCRIPTION**

全ヘッダレコード (ただし記述型レコードを除く) を調べ，SFITSIO が持つコメント辞書に該当するキーワードがあればその内容でコメント文をセット (上書き) します．このコメント辞書は，各 HDU タイプ専用の辞書が 3 つ，汎用の辞書が 1 つから構成され，どれを使うかは引数 `hdutype` で指定します．なお，それぞれの HDU 専用の辞書にキーワードが存在しない場合は，汎用のコメント辞書が使われます．

引数として指定できるのは，`FITS::IMAGE_HDU`, `FITS::BINARY_TABLE_HDU`, `FITS::ASCII_TABLE_HDU`, `FITS::ANY_HDU`，のいずれかで，`FITS::ANY_HDU` が指定された場合と引数を省略した場合は，HDU のタイプをヘッダの内容から自動判定します．

コメント辞書の変更については，§12.4.40 をご覧ください．

**PARAMETER**

[I] `hdutype` HDU のタイプ
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_hdu オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (`sli::err_rec` 例外) が発生します．

**EXAMPLES**

次のコードは，プライマリ HDU のすべての記述型でないレコードのコメント部を，コメント辞書の内容でセットします．

```
fits.image("Primary").header_assign_default_comments(FITS::IMAGE_HDU);
```

---

### 12.4.40 fits::update_comment_dictionary()

**NAME**

fits::update_comment_dictionary() — コメント辞書の内容を追加・変更する

**SYNOPSIS**

```
#include <sli/fits_header_record.h>
const asarray_tstring &update_comment_dictionary( int hdutype,
                                                  const char *const *new_kwd_comments );
```

**DESCRIPTION**

SFITSIO が持つコメント辞書の内容を変更します．コメント辞書は，各 HDU タイプ専用の辞書が 3 つ，汎用の辞書が 1 つから構成されます．汎用のコメント辞書はそれぞれの HDU 専用の辞書にキーワードが存在しない場合にも使われます．これらのコメント辞書は，hdu().header_fill_blank_comments() hdu().header_assign_default_comments() で使用されます (§12.4.38，§12.4.39を参照)．

引数 hdutype によって，どの HDU タイプ用のコメント辞書を変更するかを設定します．指定できるのは，FITS::IMAGE_HDU, FITS::BINARY_TABLE_HDU, FITS::ASCII_TABLE_HDU, FITS::ANY_HDU, のいずれかです．

引数 new_kwd_comments には更新したいキーワードとコメントのセットを，キーワード 1，コメント 1，キーワード 2，コメント 2，… ，NULL の順に文字列のポインタ配列で与えます (要 NULL 終端)．

**PARAMETER**

| | | |
|---|---|---|
| [I] | hdutype | HDU のタイプ |
| [I] | new_kwd_comments | キーワードとコメントを有する文字列のポインタ配列 |

([I]：入力，[O]：出力)

**RETURN VALUE**

コメント辞書が保存されている asarray_tstring オブジェクトの参照 (読み取りのみ) を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

次のコードは，汎用のコメント辞書とバイナリテーブル用のコメント辞書を更新します．

```
#include <sli/fits_header_record.h>

/* comment strings for blank comment in template */
static const char *Kwd_comment_any[] = {
    /*           |MIN                                          MAX| */
    "TIME-EPH", "epoch time for FITS header",
    "SMPLBASE", "base telemetry referred to shrink rows",
    /*           |MIN                                          MAX| */
    NULL
};
static const char *Kwd_comment_bintable[] = {
    /*           |MIN                                          MAX| */
    "TTNAM#",   "original telemetry name",
    "TCONV#",   "type of data conversion",
    "TSTAT#",   "definition of status values",
    "TINPL#",   "type of interpolation",
    "TSPAN#",   "[s] maximum span for interpolation",
    "", "",
    /*           |MIN                                          MAX| */
```

```
    NULL
};

int main()
{
    /* update comment dictionaries */
    fits::update_comment_dictionary(FITS::ANY_HDU, Kwd_comment_any);
    fits::update_comment_dictionary(FITS::BINARY_TABLE_HDU, Kwd_comment_bintable);
```

## 12.5 ヘッダの操作 (低レベル)・ディスクベースの FITS I/O

fits_header クラスには，オープンされたストリームに対して Header Unit の読み書き，および Data Unit の読み飛ばしを行なうためのメンバ関数が用意されています．これらを使うと，FITS ヘッダのみに対する高速アクセスや，ストリーム操作 API を使って自由度の高い Data Unit へのアクセスが可能です．

SFITSIO のソフトウェアパッケージには，FITS ヘッダに高速アクセスするためのツール「tools/hv.cc」が付属しています．このツールは，ここで取り上げる API を使って作られていますので，例として参照すると理解の助けになるでしょう．

### 12.5.1 fits_header::read_stream()

**NAME**

fits_header::read_stream() — Header Unit だけの読み取り

**SYNOPSIS**

```
ssize_t fits_header::read_stream( cstreamio &sref );
```

**DESCRIPTION**

オープン済みのストリーム sref から 1 つの Header Unit を読み取ってパースし，自身にその内容を格納します．

このメンバ関数の終了後，ストリームの位置は続く Data Unit の先頭にセットされます．

ストリーム sref は，cstreamio クラスの継承クラスのいずれかを指定してください．これらのクラスについては，APPENDIX と SLLIB のマニュアルを参照してください．

**PARAMETER**

[I] sref オープンされたストリームを扱うオブジェクト

([I]：入力，[O]：出力)

**RETURN VALUE**

| | | |
|---|---|---|
| 正の値 | : | ストリームから読み取られたバイト数． |
| ゼロ | : | FITS ファイルのストリームが終了した場合． |
| 負の値 (エラー) | : | ストリームの操作に失敗した場合． |

---

### 12.5.2 fits_header::write_stream()

**NAME**

fits_header::write_stream() — Header Unit だけの書き出し

**SYNOPSIS**

```
ssize_t fits_header::write_stream( cstreamio &sref, bool end_and_blank );
```

**DESCRIPTION**

オープン済みのストリーム sref に対し，自身の内容を FITS 規約に従った形式 (横 80 文字のテキスト) で出力します．

引数 end_and_blank が true の場合，END キーワードと空白文字によるパディングが行なわれるので，このメンバ関数の終了後にストリームに対して続けて出力する事で，Data Unit を先頭から書き込む事ができます．

引数 end_and_blank が false の場合は，END キーワード以降は出力されません．

ストリーム sref は，cstreamio クラスの継承クラスのいずれかを指定してください．これらのクラスについては，APPENDIX と SLLIB のマニュアルを参照してください．

**PARAMETER**

[I] sref オープンされたストリームを扱うオブジェクト
[I] end_and_blank END キーワードと空白文字を書き込むかどうか
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : ストリームに書き出されたバイト数．
負の値 (エラー) : ストリームの操作に失敗した場合．

---

### 12.5.3 fits_header::skip_data_stream()

**NAME**

fits_header::skip_data_stream() — 1 つの Data Unit を読み飛ばし

**SYNOPSIS**

```
ssize_t fits_header::skip_data_stream( cstreamio &sref );
```

**DESCRIPTION**

自身が保持しているヘッダ情報に基づき，オープン済みのストリーム sref において 1 つの Data Unit を読み飛ばします．このメンバ関数を使う前に，あらかじめ read_stream() メンバ関数 (§12.5.1) によりオブジェクトに FITS ヘッダの情報を与えておく必要があります．

このメンバ関数が終了した時点でのストリームの位置は，HDU が続く場合は次の Header Unit の先頭にセットされます．なお，HDU が続くかどうかは次に呼ばれる read_stream() メンバ関数 (§12.5.1) の返り値で判定します．

ストリーム sref は，cstreamio クラスの継承クラスのいずれかを指定してください．これらのクラスについては，APPENDIX と SLLIB のマニュアルを参照してください．

**PARAMETER**

[I] sref オープンされたストリームを扱うオブジェクト
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : ストリームから読み取られたバイト数．
負の値 (エラー) : ストリームの操作に失敗した場合．

---

## 12.6 Image HDU の操作

ここでは，Image HDU を操作するための API を解説します.「image( ... )」の括弧内の引数はすべて同じですので，その部分の引数に関する記述は省略します．この括弧内の引数には，HDU 番号 (long index) あるいは HDU 名 (const char *hduname) を指定します.

### 12.6.1 image().hduname(), image().assign_hduname()

**NAME**

image().assign_hduname(), image().hduname() — HDU 名の操作

**SYNOPSIS**

```
const char *image( ... ).hduname();
const char *image( ... ).extname();
fits_image &image( ... ).assign_hduname( const char *name );
fits_image &image( ... ).assign_extname( const char *name );
```

**DESCRIPTION**

image().hduname() メンバ関数は，HDU の名称を返します.

image().assign_hduname() メンバ関数は，HDU の名称を変更します．名称 name はヘッダ EXTNAME に反映されます．Primary HDU の場合も指定可能です.

**PARAMETER**

[I] name セットする HDU 名称
([I]：入力，[O]：出力)

**RETURN VALUE**

image().hduname() は HDU 名称文字列のアドレスを返します.
image().assign_hduname() は当該 fits_image オブジェクトへの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，HDU 名称変更時の領域リサイズ処理で失敗した場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
printf("HDU 名=%s\n", fits.image("Primary").hduname());
```

---

### 12.6.2 image().hduver(), image().assign_hduver()

**NAME**

image().assign_hduver(), image().hduver() — HDU のバージョン番号の操作

**SYNOPSIS**

```
long long image( ... ).hduver();
long long image( ... ).extver();
fits_image &image( ... ).assign_hduver( long long ver );
fits_image &image( ... ).assign_extver( long long ver );
```

**DESCRIPTION**

image().hduver() メンバ関数は，HDU のバージョン番号を返します.

image().assign_hduver() メンバ関数は，HDU のバージョン番号を変更します．名称 ver はヘッダ EXTVER に反映されます．Primary HDU の場合も指定可能です.

**PARAMETER**

[I] ver セットするバージョン番号

([I]：入力，[O]：出力)

**RETURN VALUE**

image().hduver() は HDU のバージョン番号を返します.

image().assign_hduver() は当該 fits_image オブジェクトへの参照を返します.

**EXAMPLES**

```
printf("HDU バージョン=%lld\n", fits.image("Primary").hduver());
```

---

### 12.6.3 image().dim_length()

**NAME**

image().dim_length() — 座標軸の数

**SYNOPSIS**

```
long image( ... ).dim_length() const;
long image( ... ).axis_length() const;
```

**RETURN VALUE**

座標軸の数を返します.

**EXAMPLES**

```
printf("次元数=%ld\n", fits.image("Primary").dim_length());
```

---

### 12.6.4 image().length()

**NAME**

image().length() — ピクセル数

**SYNOPSIS**

```
long image( ... ).length() const;
long image( ... ).length( long axis ) const;
```

**DESCRIPTION**

軸 axis のピクセル数を返します．axis が省略された時は，全ピクセル数を返します.

**PARAMETER**

[I] axis ピクセル数を取得する座標軸

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル数を返します.

**EXAMPLES**

次のコードは ImageHDU の総ピクセル数と各座標軸のピクセル数を表示します.

```
long axis;
printf("総ピクセル数=%ld\n", fits.image("Primary").length());
for ( axis = 0 ; axis < fits.image("Primary").dim_length() ; axis++ ) {
    printf("座標軸 [%ld] ピクセル数 [%ld]\n",
           axis, fits.image("Primary").length(axis));
}
```

---

### 12.6.5 image().type()

**NAME**

image().type() — ピクセル値の型

**SYNOPSIS**

```
int image( ... ).type() const;
```

**RETURN VALUE**

ピクセル値の型を返します. 返される値は, FITS::DOUBLE_T, FITS::FLOAT_T, FITS::LONGLONG_T, FITS::LONG_T, FITS::SHORT_T, FITS::BYTE_T のいずれかです.

**EXAMPLES**

次のコードはピクセル値のデータ型を調べています.

```
switch ( fits.image("Primary").type() ) {
  case FITS::DOUBLE_T:
      printf("DOUBLE です\n");
      break;
  case FITS::FLOAT_T:
      printf("FLOAT です\n");
      break;
  case FITS::LONGLONG_T:
      printf("LONGLONG です\n");
      break;
  case FITS::LONG_T:
      printf("LONG です\n");
      break;
  case FITS::SHORT_T:
      printf("SHORT です\n");
      break;
  case FITS::BYTE_T:
      printf("BYTE です\n");
```

```
        break;
    default:
        printf("不明なレコード型です\n");
        break;
}
```

---

### 12.6.6 image().bytes()

**NAME**

image().bytes() — 1 ピクセルのバイトサイズ

**SYNOPSIS**

```
long image( ... ).bytes() const;
```

**RETURN VALUE**

1 ピクセルのバイトサイズを返します.

**EXAMPLES**

```
printf("1ピクセルのバイトサイズは%ldです\n", fits.image("Primary").bytes());
```

---

### 12.6.7 image().col_length()

**NAME**

image().col_length() — カラム ($x$ 軸) のピクセル数

**SYNOPSIS**

```
long image( ... ).col_length() const;
```

**RETURN VALUE**

カラム (axis0; $x$ 軸) のピクセル数を返します.

**EXAMPLES**

```
printf("X軸のピクセル数は %ld です\n", fits.image("Primary").col_length());
```

---

### 12.6.8 image().row_length()

**NAME**

image().row_length() — ロウ ($y$ 軸) のピクセル数

**SYNOPSIS**

```
long image( ... ).row_length() const;
```

**RETURN VALUE**

ロウ (axis1; $y$ 軸) のピクセル数を返します.

**EXAMPLES**

```
printf("Y軸のピクセル数は %ld です\n", fits.image("Primary").row_length());
```

---

### 12.6.9 image().layer_length()

**NAME**

image().layer_length() — レイヤ ($z$ 軸) のピクセル数

**SYNOPSIS**

```
long image( ... ).layer_length() const;
```

**RETURN VALUE**

レイヤ ($z$ 軸) のピクセル数を返します．次元数が 3 を越える場合は，axis2 以降すべてが対象です．

**EXAMPLES**

```
printf("Z軸以上の総ピクセル数は %ld です\n", fits.image("Primary").layer_length());
```

---

### 12.6.10 image().dvalue()

**NAME**

image().dvalue() — 実数のピクセル値を返す

**SYNOPSIS**

```
double image( ... ).dvalue( long axis0, long axis1 = FITS::INDEF,
                            long axis2 = FITS::INDEF ) const;
double image( ... ).dvalue_v( long num_axisx,
                              long axis0, long axis1, long axis2, ... ) const;
double image( ... ).va_dvalue_v( long num_axisx, long axis0, long axis1, long axis2,
                                 va_list ap ) const;
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 のピクセル値を返します．このメンバ関数で返される値はヘッダの BZERO，BSCALE，BLANK を反映した値です．BLANK 値の場合や，引数の座標値が範囲外の場合，NAN を返します．

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます．EXAMPLES では，$n$ 次元のデータを 1 次元として扱っています．

ユーザは定数 FITS::INDEF を明示的に使わないでください．

**PARAMETER**

| | | |
|---|---|---|
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します．

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，全ピクセル値の総和を求めています.

```
double sum = 0;
for ( i=0 ; i < fits.image("Primary").length() ; i++ ) {
    sum += fits.image("Primary").dvalue(i);
}
```

チュートリアルの §5.8にも使用例があります.

---

### 12.6.11 image().lvalue(), image().llvalue()

**NAME**

image().lvalue(), image().llvalue() — 整数のピクセル値を返す

**SYNOPSIS**

```
long image( ... ).lvalue( long axis0, long axis1 = FITS::INDEF,
                          long axis2 = FITS::INDEF ) const;
long image( ... ).lvalue_v( long num_axisx,
                            long axis0, long axis1, long axis2, ... ) const;
long image( ... ).va_lvalue_v( long num_axisx, long axis0, long axis1, long axis2,
                               va_list ap ) const;
long long image( ... ).llvalue( long axis0, long axis1 = FITS::INDEF,
                                long axis2 = FITS::INDEF ) const;
long long image( ... ).llvalue_v( long num_axisx,
                                  long axis0, long axis1, long axis2, ... ) const;
long long image( ... ).va_llvalue_v( long num_axisx,
                                     long axis0, long axis1, long axis2,
                                     va_list ap ) const;
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 のピクセル値を返します．このメンバ関数で返される値はヘッダの BZERO，BSCALE，BLANK を反映した値です．BLANK 値の場合や，引数の座標値が範囲外の場合，INDEF_LONG または INDEF_LLONG を返します.

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.6.10の EXAMPLES を参照してください．チュートリアルの §5.8にも使用例があります.

---

### 12.6.12 image().assign()

**NAME**

image().assign() — ピクセル値を変更する

**SYNOPSIS**

```
fits_image &image( ... ).assign( double value, long axis0,
                                 long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
fits_image &image( ... ).assign_v( double value, long num_axisx,
                                   long axis0, long axis1, long axis2, ... );
fits_image &image( ... ).va_assign_v( double value, long num_axisx,
                                      long axis0, long axis1, long axis2,
                                      va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 のピクセル値を変更します．このメンバ関数では，ヘッダの BZERO，BSCALE を反映した値で内部バッファのピクセル値を更新します．value に NAN を与えた場合，型が整数型の場合に BLANK 値が存在すれば，BLANK 値を内部バッファに書き込みます．BLANK 値が存在しない場合，型に応じて INDEF_UCHAR，INDEF_INT16，INDEF_INT32，INDEF_INT64 が書き込まれます.

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

[I] value セットするピクセル値
[I] axis0 カラム ($x$ 軸) ピクセル位置の指定
[I] axis1 ロウ ($y$ 軸) ピクセル位置の指定
[I] axis2 レイヤ ($z$ 軸) ピクセル位置の指定
[I] num_axisx 引数で指定している次元軸の数
[I] ... 各次元内のピクセル位置全要素データ
[I] ap 各次元内のピクセル位置全要素データ
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは 0 行目の全てのピクセル値を 0 に変更しています.

```
double value = 0;
for ( i=0 ; i < fits.image("Primary").col_length() ; i++ ) {
    fits.image("Primary").assign(value, i,0);
}
```

チュートリアルの §5.8にも使用例があります.

---

### 12.6.13 image().convert_type()

**NAME**

image().convert_type() — データ型の変換・変更

**SYNOPSIS**

```
fits_image &image( ... ).convert_type( int new_type );
fits_image &image( ... ).convert_type( int new_type, double new_zero );
fits_image &image( ... ).convert_type( int new_type, double new_zero,
                                       double new_scale );
fits_image &image( ... ).convert_type( int new_type, double new_zero,
                                       double new_scale, long long new_blank );
```

**DESCRIPTION**

イメージのデータ型を new_type に変換します．必要に応じて，内部バッファのサイズも変更します．new_type に指定できる値は，FITS::DOUBLE_T，FITS::FLOAT_T，FITS::LONGLONG_T，FITS::LONG_T，FITS::SHORT_T，FITS::BYTE_T のいずれかです．new_zero，new_scale，new_blank を指定した場合は，ヘッダの BZERO，BSCALE，BLANK も変更し，それらの値を反映したイメージデータに変換します．new_blank が有効なのは，new_type が整数型の場合のみです.

**PARAMETER**

[I] `new_type` 変換後のデータ型
[I] `new_zero` 変換後の BZERO 値
[I] `new_scale` 変換後の BSCALE 値
[I] `new_blank` 変換後の BLANK 値
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，変換後にイメージデータが拡大される場合)，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードは，Primary HDU のあらゆるタイプの画像データを double 型に変換します.

```
fits.image("Primary").convert_type(FITS::DOUBLE_T);
```

チュートリアルの §5.11 にも使用例があります.

---

### 12.6.14 image().bzero(), image().assign_bzero()

**NAME**

image().bzero()，image().assign_bzero() — ゼロ点の操作

**SYNOPSIS**

```
double image( ... ).bzero() const;
bool image( ... ).bzero_is_set() const;
fits_image &image( ... ).assign_bzero( double zero, int prec = 15 );
fits_image &image( ... ).erase_bzero();
```

**DESCRIPTION**

image().bzero() メンバ関数は，ヘッダの BZERO の値を返します．image().bzero_is_set() メンバ関数は，ヘッダの BZERO の有無を返します.

image().assign_bzero() メンバ関数は，ヘッダの BZERO の値を設定します．`prec` には桁数を指定できます．省略した場合，15 桁の精度でヘッダレコードに書き込みます.

image().erase_bzero() メンバ関数は，ヘッダの BZERO を消去します.

これらのメンバ関数ではイメージの実際のゼロ点は変化しません．イメージも同時に変更したい場合は，`image().convert_type()`(§12.6.13) を使います.

**PARAMETER**

[I] `zero` セットする BZERO 値
[I] `prec` 精度 (桁数)
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，image().assign_bzero() メンバ関数内でアドレステーブル領域の再確保で失敗した場合)，SLLIB またはSFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードは BZERO の有無を確認し，無い場合は値を設定しています.

```
if ( fits.image("Primary").bzero_is_set() == false ) {
    fits.image("Primary").assign_bzero(0.0);
}
```

---

### 12.6.15 image().bscale(), image().assign_bscale()

**NAME**

image().bscale(), image().assign_bscale() — スケーリングファクターの操作

**SYNOPSIS**

```
double image( ... ).bscale() const;
bool image( ... ).bscale_is_set() const;
fits_image &image( ... ).assign_bscale( double scale, int prec = 15 );
fits_image &image( ... ).erase_bscale();
```

**DESCRIPTION**

image().bscale() メンバ関数は，ヘッダの `BSCALE` の値を返します．image().bscale_is_set() メンバ関数は，ヘッダの `BSCALE` の有無を返します.

image().assign_bscale() メンバ関数は，ヘッダの `BSCALE` の値を設定します．`prec` には桁数を指定できます．省略した場合，15 桁の精度でヘッダレコードに書き込みます.

image().erase_bscale() メンバ関数は，ヘッダの `BSCALE` を消去します.

これらのメンバ関数ではイメージの実際のスケーリングファクターは変化しません．イメージも同時に変更したい場合は，`image().convert_type()`(§12.6.13) を使います.

**PARAMETER**

[I] `scale` セットする `BSCALE` 値
[I] `prec` 精度 (桁数)
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，image().assign_bscale() メンバ関数内でアドレステーブル領域の再確保で失敗した場合)，SLLIB またはSFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードは BSCALE の有無を確認し，無い場合は値を設定しています.

```
if ( fits.image("Primary").bscale_is_set() == false ) {
    fits.image("Primary").assign_bscale(1.0);
}
```

---

### 12.6.16 image().blank(), image().assign_blank()

**NAME**

image().blank() — ブランク値の操作

**SYNOPSIS**

```
long long image( ... ).blank() const;
bool image( ... ).blank_is_set() const;
fits_image &image( ... ).assign_blank( long long blank );
fits_image &image( ... ).erase_blank();
```

**DESCRIPTION**

image().blank() メンバ関数は，ヘッダの BLANK の値を返します．image().blank_is_set() メンバ関数は，ヘッダの BLANK の有無を返します．

image().assign_blank() メンバ関数は，ヘッダの BLANK の値を設定します．image().erase_blank() メンバ関数は，ヘッダの BLANK を消去します．

**PARAMETER**

[I] `blank` セットする BLANK 値

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，image().assign_blank() メンバ関数内でアドレステーブル領域の再確保で失敗した場合)，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します．

**EXAMPLES**

次のコードは BlANK の有無を確認し，無い場合は値を設定しています．

```
if ( fits.image("Primary").blank_is_set() == false ) {
    fits.image("Primary").assign_blank(0);
}
```

---

### 12.6.17 image().bunit(), image().assign_bunit()

**NAME**

image().bunit(), image().assign_bunit() — 物理単位の操作

**SYNOPSIS**

```
const char *image( ... ).bunit();
bool image( ... ).bunit_is_set();
fits_image &image( ... ).assign_bunit( const char *unit );
fits_image &image( ... ).erase_bunit();
```

**DESCRIPTION**

image().bunit() メンバ関数は，ヘッダの BUNIT の値 (物理単位の文字列) を返します．返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，ヘッダの BUNIT 値が変更された場合は無効になります．

image().bunit_is_set() メンバ関数は，ヘッダの BUNIT の有無を返します．

image().assign_bunit() メンバ関数は，ヘッダの BUNIT の値 (物理単位) を設定します．image().erase_bunit() メンバ関数は，ヘッダの BUNIT を消去します．

**PARAMETER**

[I] `unit` セットする BUNIT 文字列 (アドレス)
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，image().assign_bunit() メンバ関数内でアドレステーブル領域の再確保で失敗した場合)，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します．

**EXAMPLES**

次のコードは BUNIT の有無を確認し，無い場合は値を設定しています．

```
if ( fits.image("Primary").bunit_is_set() == false ) {
    fits.image("Primary").assign_bunit("ADU");
}
```

---

### 12.6.18 image().init()

**NAME**

image().init() — イメージとヘッダの初期化

**SYNOPSIS**

```
fits_image &image( ... ).init();
fits_image &image( ... ).init( const fits_image &obj );
fits_image &image( ... ).init( int type,
                               long naxis0, long naxis1 = 0, long naxis2 = 0 );
fits_image &image( ... ).init( int type, long naxis, long naxisx[] );
```

**DESCRIPTION**

イメージとヘッダを初期化します．obj が指定された場合は，その内容をコピーします．

type には，イメージのデータ型を指定します．指定できるデータ型は，FITS::DOUBLE_T, FITS::FLOAT_T, FITS::LONGLONG_T, FITS::LONG_T, FITS::SHORT_T, FITS::BYTE_T のいずれかです．

naxis0, naxis1, naxis2 にはイメージのサイズとレイヤの数を指定します．

EXTNAME と EXTVER の内容は変更されません．

**PARAMETER**

[I] obj 初期化時後にコピー元となる Image オブジェクト
[I] type イメージのデータ型
[I] naxis0 カラム ($x$ 軸) のピクセル数
[I] naxis1 ロウ ($y$ 軸) のピクセル数
[I] naxis2 レイヤ ($z$ 軸) のピクセル数
[I] naxis naxisx で指定しているリスト数
[I] naxisx 各次元軸のピクセル数リスト
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，メモリの空き容量に対して初期化後のイメージデータが大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

次のコードは，Primary HDU をピクセルのデータ型を double，$x$ 軸, $y$ 軸, $z$ 軸のピクセル数をそれぞれ 100, 200, 3 で初期化しています．

```
fits.image("Primary").init(FITS::DOUBLE_T, 100, 200, 3);
```

---

### 12.6.19 image().swap()

**NAME**

image().swap() — イメージの交換

**SYNOPSIS**

```
fits_image &image( ... ).swap( fits_image &obj );
```

**DESCRIPTION**

自身と obj との中身を交換します．

**PARAMETER**

[I/O] obj 入れ替え対象のオブジェクト
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits1.image("Primary").swap( fits2.image("Primary") );
```

チュートリアルの §5.10 にも使用例があります.

---

### 12.6.20 image().increase_dim()

**NAME**

image().increase_dim() — 軸の追加

**SYNOPSIS**

```
fits_image &image( ... ).increase_dim();
fits_image &image( ... ).increase_axis();
```

**DESCRIPTION**

軸を 1 つ追加します．追加した軸のピクセル数の初期値は 1 です．ピクセル数を増やすには，`image().resize()` を使います.

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部メモリ領域確保の操作に失敗した場合，SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.image("Primary").increase_dim();
```

---

### 12.6.21 image().decrease_dim()

**NAME**

image().decrease_dim() — 軸を減らす.

**SYNOPSIS**

```
fits_image &image( ... ).decrease_dim();
fits_image &image( ... ).decrease_axis();
```

**DESCRIPTION**

軸を 1 つ減らします．最も大きな次元の軸のピクセル数が 1 ではない場合，内部バッファのサイズも同時に変更します.

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部メモリ領域確保の操作に失敗した場合，SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.image("Primary").decrease_dim();
```

---

### 12.6.22 image().resize()

**NAME**

image().resize() — ピクセル数を変更する

**SYNOPSIS**

```
fits_image &image( ... ).resize( long axis, long size );
```

**DESCRIPTION**

軸 `axis` のピクセル数を `size` に変更します．ピクセル数を増やした場合，増えた部分はヘッダの `BZERO`，`BSCALE` の値を反映したゼロ値で初期化されます.

**PARAMETER**

[I] `axis` 変更対象となる次元軸
[I] `size` 変更後のピクセル数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部メモリ領域確保の操作に失敗した場合，SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
printf("変更前のピクセル数=%ld\n", fits.image("Primary").col_length());
fits.image("Primary").resize(0, 2000);
printf("変更後のピクセル数=%ld\n", fits.image("Primary").col_length());
```

---

### 12.6.23 image().assign_default()

**NAME**

image().assign_default() — 要素長を大きくした場合の新規ピクセルの値を設定

**SYNOPSIS**

```
fits_image &image( ... ).assign_default( double value );
fits_image &image( ... ).assign_default_value( const void *value_ptr );
```

**DESCRIPTION**

.resize() 等で画像データの要素長を大きくした場合の新規ピクセルのデフォルト値を設定します.

.assign_default() は高レベルなメンバ関数で，ヘッダの BZERO，BSCALE，BLANK の値が反映されます．BLANK 値をセットしたい場合は NAN をセットします.

.assign_default_value() は低レベルなメンバ関数で，ヘッダの BZERO 等の値は考慮されません．オブジェクト内の画像の型に一致する変数または定数のアドレスを与える必要があります.

**PARAMETER**

[I] value デフォルト値
[I] value_ptr デフォルト値を持つユーザ変数または定数のアドレス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部メモリ領域確保の操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードでは，リサイズした時の新規ピクセルの値を BLANK 値に設定し，画像の $x$ 方向をリサイズします.

```
fits.image("Primary").assign_default(NAN).resize(0, 2000);
```

---

### 12.6.24 image().fix_rect_args()

**NAME**

image().fix_rect_args() — 矩形領域を指定するための引数の値のチェックと修正

**SYNOPSIS**

```
int image( ... ).fix_rect_args( long *r_col_index, long *r_col_size,
                                long *r_row_index, long *r_row_size,
                                long *r_layer_index, long *r_layer_size ) const;
```

**DESCRIPTION**

引数で指定された矩形領域が，オブジェクトが持つ画像の範囲内にあるかどうかを調べ，範囲外の領域を指していた場合は範囲内だけを指すように引数の値を修正します.

.scan_cols() 等 (§12.6.25〜) を呼ぶ前に，この関数を使うと，処理の対象となる正確な位置と領域を得る事ができます．プログラマ側でワーク用バッファを利用する場合も，このメンバ関数を使うとワーク用バッファの正確なサイズを求める事ができます.

**PARAMETER**

| | | |
|---|---|---|
| [I/O] | `r_col_index` | カラム ($x$ 軸) ピクセル位置のポインタ |
| [I/O] | `r_col_size` | `*r_col_index` からのピクセル数のポインタ |
| [I/O] | `r_row_index` | ロウ ($y$ 軸) ピクセル位置のポインタ |
| [I/O] | `r_row_size` | `*r_row_index` からのピクセル数のポインタ |
| [I/O] | `r_layer_index` | レイヤ ($z$ 軸) ピクセル位置のポインタ |
| [I/O] | `r_layer_size` | `*r_layer_index` からのピクセル数のポインタ |

([I]：入力，[O]：出力)

**RETURN VALUE**

| | | |
|---|---|---|
| 0 | ： | 引数が自身の配列の領域内を示していた場合. |
| 正の値 | ： | 領域からはみ出しているが有効領域が存在し，引数の値が修正された場合. |
| 負の値 | ： | 有効領域が無い場合. |

---

### 12.6.25 image().scan_cols()

**NAME**

image().scan_cols() — 画像の指定領域をユーザ関数により横方向行単位でスキャン

**SYNOPSIS**

```
long image( ... ).scan_cols(
        long (*func)(double [],long, long,long,long,const fits_image *,void *),
                    void *user_ptr,
                    long col_index = 0, long col_size = FITS::ALL,
                    long row_index = 0, long row_size = FITS::ALL,
                    long layer_index = 0, long layer_size = FITS::ALL ) const;
```

**DESCRIPTION**

次の動作により，3 番目以降の引数によって指定された矩形領域をスキャンします.

(1) 指定領域の横 1 行分の double 型のテンポラリバッファを確保する.

(2) テンポラリバッファに指定領域の 1 行分のピクセル値を double 型に変換して格納する．この時，SCALE，ZERO，BLANK の変換も適切に行なわれる.

(3) ユーザ関数 `func` が呼び出される.

(4) 矩形領域内のすべての行について，(2)，(3) を行なう.

(3) のユーザ関数の中では，1 行分のピクセルをスキャンするためのループを組む必要があります．また，ユーザ関数では，1 行分のデータについての有効なピクセル数を返すようにします.

ユーザ関数 (`*func`) の引数の仕様は次のとおりです.

`double pix[]` … 自身の要素の 1 行分の配列 (テンポラリバッファ)
`long n_pix` … 配列 `pix` の個数
`long axis0` … $x$ 軸の座標
`long axis1` … $y$ 軸の座標
`long axis2` … $z$ 軸の座標

`fits_image *thisp` … 自身のオブジェクトのアドレス
`void *ptr` … `user_ptr` の値

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

プログラマは定数 `FITS::ALL` を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `func` | ユーザ関数のアドレス |
| [I] | `user_ptr` | ユーザ関数に与えられる任意のポインタ変数 |
| [I] | `col_index` | カラム ($x$ 軸) ピクセル位置 |
| [I] | `col_size` | `col_index` からのピクセル数 |
| [I] | `row_index` | ロウ ($y$ 軸) ピクセル位置 |
| [I] | `row_size` | `row_index` からのピクセル数 |
| [I] | `layer_index` | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | `layer_size` | `layer_index` からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

| | | |
|---|---|---|
| 非負の値 | : | スキャンによる有効ピクセルの数 (ユーザ関数によって返された整数の総和). |
| 負の値 (エラー) | : | 関数ポインタや引数の座標値が不正等の場合. |

**EXAMPLES**

次のコードでは，Primary HDU の画像をスキャンし，有効ピクセルの値の総和を `res.sum` に，有効ピクセルの数を `res.npix` に得ます.

```
struct pixels_results {
    double sum;
    long npix;
};

static long pixels_sum( double vals[],long n, long ii,long jj,long kk,
                        const fits_image *this_p, void *_p )
{
    struct pixels_results *resp = (struct pixels_results *)_p;
    long i, cnt = 0;
    for ( i=0 ; i < n ; i++ ) {
        if ( isfinite(vals[i]) ) {
            resp->sum += vals[i];
            cnt ++;
        }
    }
    return cnt;
}

int main()
```

```
{
    struct pixels_results res;
    :
    res.npix = fits.image("Primary").scan_cols( &pixels_sum, (void *)&res );
```

---

### 12.6.26 image().scan_rows()

**NAME**

image().scan_rows() — 画像の指定領域をユーザ関数により縦方向列単位でスキャン

**SYNOPSIS**

```
long image( ... ).scan_rows(
        long (*func)(double [],long, long,long,long,const fits_image *,void *),
                    void *user_ptr,
                    long col_index = 0, long col_size = FITS::ALL,
                    long row_index = 0, long row_size = FITS::ALL,
                    long layer_index = 0, long layer_size = FITS::ALL ) const;
```

**DESCRIPTION**

次の動作により，3 番目以降の引数によって指定された矩形領域をスキャンします.

(1) 指定領域の縦 1 列分の double 型のテンポラリバッファを確保する.

(2) テンポラリバッファに指定領域の 1 列分のピクセル値を double 型に変換して格納する．この時，SCALE，ZERO，BLANK の変換も適切に行なわれる.

(3) ユーザ関数 func が呼び出される.

(4) 矩形領域内のすべての列について，(2)，(3) を行なう.

(3) のユーザ関数の中では，縦 1 列分のピクセルをスキャンするためのループを組む必要があります．また，ユーザ関数では，1 列分のデータについての有効なピクセル数を返すようにします.

ユーザ関数 (*func) の引数の仕様は次のとおりです.

- double pix[] … 自身の要素の 1 列分の配列 (テンポラリバッファ)
- long n_pix … 配列 pix の個数
- long axis0 … $x$ 軸の座標
- long axis1 … $y$ 軸の座標
- long axis2 … $z$ 軸の座標
- fits_image *thisp … 自身のオブジェクトのアドレス
- void *ptr … user_ptr の値

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

プログラマは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `func` | ユーザ関数のアドレス |
| [I] | `user_ptr` | ユーザ関数に与えられる任意のポインタ変数 |
| [I] | `col_index` | カラム ($x$ 軸) ピクセル位置 |
| [I] | `col_size` | `col_index` からのピクセル数 |
| [I] | `row_index` | ロウ ($y$ 軸) ピクセル位置 |
| [I] | `row_size` | `row_index` からのピクセル数 |
| [I] | `layer_index` | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | `layer_size` | `layer_index` からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : スキャンによる有効ピクセルの数 (ユーザ関数によって返された整数の総和).

負の値 (エラー) : 関数ポインタや引数の座標値が不正等の場合.

**EXAMPLES**

§12.6.25の EXAMPLES をご覧ください.

---

### 12.6.27 image().scan_layers()

**NAME**

image().scan_layers() — 指定領域をユーザ関数により $z$ 方向にピクセル単位でスキャン

**SYNOPSIS**

```
long image( ... ).scan_layers(
   long (*func)(double [],long,long, long,long,long,const fits_image *,void *),
                    void *user_ptr,
                    long col_index = 0, long col_size = FITS::ALL,
                    long row_index = 0, long row_size = FITS::ALL,
                    long layer_index = 0, long layer_size = FITS::ALL ) const;
```

**DESCRIPTION**

次の動作により，3 番目以降の引数によって指定された矩形領域をスキャンします.

(1) 指定領域の「$z$ 方向の長さ×横 1 行」分の double 型のテンポラリバッファを確保する.

(2) テンポラリバッファに指定領域の「$z$ 方向の長さ×横 1 行」分のピクセル値を double 型に変換して格納する．この時，SCALE，ZERO，BLANK の変換も適切に行なわれる.

(3) ユーザ関数 `func` が呼び出される.

(4) 矩形領域内のすべての行について，(2)，(3) を行なう.

(3) のユーザ関数の中では，$z$ 方向および横 1 行分のピクセルをスキャンするためのループを組む必要があります (内側のループで $z$ 方向をスキャンします)．また，ユーザ関数では，1 行分のデータについての有効なピクセル数を返すようにします.

ユーザ関数 (`*func`) の引数の仕様は次のとおりです.

`double pix[]` … 自身の要素の「$z$ 方向の長さ×横 1 行」分の配列 (テンポラリバッファ)

long n_zpix … 配列 pix の $z$ 方向の長さ
long n_xpix … 配列 pix の横方向の長さ
long axis0 … $x$ 軸の座標
long axis1 … $y$ 軸の座標
long axis2 … $z$ 軸の座標
fits_image *thisp … 自身のオブジェクトのアドレス
void *ptr … user_ptr の値

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

プログラマは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | func | ユーザ関数のアドレス |
| [I] | user_ptr | ユーザ関数に与えられる任意のポインタ変数 |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

| | | |
|---|---|---|
| 非負の値 | : | スキャンによる有効ピクセルの数 (ユーザ関数によって返された整数の総和). |
| 負の値 (エラー) | : | 関数ポインタや引数の座標値が不正等の場合. |

**EXAMPLES**

次のコードでは，Primary HDU の画像をスキャンし，ピクセルごとに $z$ 方向すべての有効ピクセル値の総和を求め，その結果をバッファに格納します.

```
static long combine_sum( double pix[], long nz, long nx,
                         long x, long y, long z,
                         const fits_image *this_p, void *_out_buf_p )
{
    double **out_buf_p = (double **)_out_buf_p;
    double *out_buf = *out_buf_p;
    double v;
    long i,j, nz_valid, nx_valid = 0, ix = 0;
    for ( i=0 ; i < nx ; i++ ) {                /* loop for X */
        v = 0;
        nz_valid = 0;
        for ( j=0 ; j < nz ; j++ ) {            /* loop for Z */
            if ( isfinite(pix[ix]) ) {          /* if valid */
                v += pix[ix];
                nz_valid ++;
```

```
            }
            ix ++;                                  /* count always */
        }
        if ( 0 < nz_valid ) {
            out_buf[i] = v;                         /* save the result */
            nx_valid ++;
        }
        else out_buf[i] = NAN;
    }
    *out_buf_p = out_buf + nx;                      /* set next position */
    return nx_valid;
}

int main()
{
    double *buf;
    :
    (buf に必要な領域を確保)
    :
    double *t_buf = buf;
    res.npix = fits.image("Primary").scan_layers( &combine_sum,
                                                  (void *)(&t_buf) );
```

---

### 12.6.28 image().fill()

**NAME**

image().fill() — 矩形領域のピクセル値を 1 つの値に変更

**SYNOPSIS**

```
fits_image &image( ... ).fill( double value,
                               long col_index = 0, long col_size = FITS::ALL,
                               long row_index = 0, long row_size = FITS::ALL,
                               long layer_index = 0, long layer_size = FITS::ALL );
```

**DESCRIPTION**

2 番目以降の引数によって指定された領域を value に変更します.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

ユーザは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

[I] value セットするピクセル値
[I] col_index カラム ($x$ 軸) ピクセル位置
[I] col_size col_index からのピクセル数
[I] row_index ロウ ($y$ 軸) ピクセル位置
[I] row_size row_index からのピクセル数
[I] layer_index レイヤ ($z$ 軸) ピクセル位置
[I] layer_size layer_index からのピクセル数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES**

次のコードでは，全てのピクセルに値 1 が設定されます.

```
fits.image("Primary").fill(1);
```

---

### 12.6.29 image().add()

**NAME**

image().add() — 矩形領域のピクセル値を 1 つの値で加算

**SYNOPSIS**

```
fits_image &image( ... ).add( double value,
                              long col_index = 0, long col_size = FITS::ALL,
                              long row_index = 0, long row_size = FITS::ALL,
                              long layer_index = 0, long layer_size = FITS::ALL );
```

**DESCRIPTION**

2 番目以降の引数によって指定された領域のピクセル値に value を加算します.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

ユーザは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

[I] value 加算する値
[I] col_index カラム ($x$ 軸) ピクセル位置
[I] col_size col_index からのピクセル数
[I] row_index ロウ ($y$ 軸) ピクセル位置
[I] row_size row_index からのピクセル数
[I] layer_index レイヤ ($z$ 軸) ピクセル位置
[I] layer_size layer_index からのピクセル数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES** 次のコードでは，全てのピクセル値に 1 が加算されます.

```
fits.image("Primary").add(1);
```

---

### 12.6.30 image().subtract()

**NAME**

image().subtract() — 矩形領域のピクセル値を 1 つの値で減算

**SYNOPSIS**

```
fits_image &image( ... ).subtract( double value,
                                   long col_index = 0, long col_size = FITS::ALL,
                                   long row_index = 0, long row_size = FITS::ALL,
                                   long layer_index = 0, long layer_size = FITS::ALL );
```

**DESCRIPTION**

2 番目以降の引数によって指定された領域のピクセル値から value を減算します.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

ユーザは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 減算する値 |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES** 次のコードでは，全てのピクセル値から 1.0 が減算されます.

```
fits.image("Primary").subtract(1.0);
```

---

### 12.6.31 image().multiply()

**NAME**

image().multiply() — 矩形領域のピクセル値を 1 つの値で乗算

**SYNOPSIS**

```
fits_image &image( ... ).multiply( double value,
                                   long col_index = 0, long col_size = FITS::ALL,
```

```
                                long row_index = 0, long row_size = FITS::ALL,
                                long layer_index = 0, long layer_size = FITS::ALL );
```

**DESCRIPTION**

2 番目以降の引数によって指定された領域のピクセル値に value を乗算します.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

ユーザは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 乗算する値 |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES**

次のコードでは，全てのピクセル値が倍になります.

```
fits.image("Primary").multiply(2);
```

---

### 12.6.32 image().divide()

**NAME**

image().divide() — 矩形領域のピクセル値を 1 つの値で除算

**SYNOPSIS**

```
fits_image &image( ... ).divide( double value,
                                long col_index = 0, long col_size = FITS::ALL,
                                long row_index = 0, long row_size = FITS::ALL,
                                long layer_index = 0, long layer_size = FITS::ALL );
```

**DESCRIPTION**

2 番目以降の引数によって指定された領域のピクセル値を value で除算します.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします.

ユーザは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 除算する値 |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES**

次のコードでは，全てのピクセル値を 3.0 で除算します.

```
fits.image("Primary").divide(3.0);
```

---

### 12.6.33 image().fill()

**NAME**

image().fill() — 矩形領域のピクセル値をユーザ関数経由で変更

**SYNOPSIS**

```
fits_image &image( ... ).fill( double value,
       void (*func)(double [],double,long, long,long,long,fits_image *,void *),
       void *user_ptr,
       long col_index = 0, long col_size = FITS::ALL,
       long row_index = 0, long row_size = FITS::ALL,
       long layer_index = 0, long layer_size = FITS::ALL );
```

**DESCRIPTION**

4 番目以降の引数によって指定された領域のピクセル値を，ユーザ関数 (*func) 経由で変更します.

このメンバ関数では，ユーザ関数でのデータ入出力用のテンポラリバッファが用意され，指定領域内で 1 行ごとのラスタースキャン (横方向行単位でのスキャン) が行なわれて，それぞれの行で，(1) テンポラリバッファにオブジェクトのピクセルバッファの内容を変換・コピー，(2) ユーザ関数の呼び出し，(3) テンポラリバッファの内容をオブジェクトのピクセルバッファへ変換・コピー，が行なわれます．この時，SCALE，ZERO，BLANK の変換も適切に行なわれます．なお，ユーザ関数の中では，横 1 列分のピクセルを変更するためのループを組む必要があります.

ユーザ関数 (*func) の引数の仕様は次のとおりで，変更結果を pix[] に対して書き込むようにコードを組んでください.

double pix[] … テンポラリバッファ上の元のピクセル値
double value … image().fill() の最初の引数

long n_pix … pix の個数
long axis0 … $x$ 軸の座標
long axis1 … $y$ 軸の座標
long axis2 … $z$ 軸の座標
fits_image *thisp … 自身のオブジェクトのアドレス
void *ptr … user_ptr の値

user_ptr はユーザが自由に使えるポインタ変数で，この値はユーザ関数の最後の引数に与えられます.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします．ただし，ユーザは定数 FITS::ALL を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | セットに用いるピクセル値 |
| [I] | func | ユーザ関数のアドレス |
| [I] | user_ptr | ユーザ関数に与えられる任意のポインタ変数 |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES**

thresh を越えるピクセル値は一定の値にしてしまう例です.

```
static void myfunc( double pix[], double value, long n_pix,
                    long axis0, long axis1, long axis2,
                    fits_image *thisp, void *ptr )
{
    long i;
    for ( i=0 ; i < n_pix ; i++ ) {
        if ( value < pix[i] ) pix[i] = value;
    }
    return;
}
    :
    :
    fits_image &primary = fits.image("Primary");
    primary.fill( thresh, &myfunc, NULL,
                  0, primary.col_length(), 0, primary.row_length() );
```

### 12.6.34 image().copy(), image().cut()

**NAME**

image().copy(), image().cut() — 矩形領域のコピー・カット

**SYNOPSIS**

```
void image( ... ).copy( fits_image *dest_img ) const;
void image( ... ).copy( fits_image *dest_img,
                        long col_index, long col_size = FITS::ALL,
                        long row_index = 0, long row_size = FITS::ALL,
                        long layer_index = 0, long layer_size = FITS::ALL ) const;
fits_image &image( ... ).cut( fits_image *dest_img,
                              long col_index, long col_size = FITS::ALL,
                              long row_index = 0, long row_size = FITS::ALL,
                              long layer_index = 0, long layer_size = FITS::ALL );
```

**DESCRIPTION**

レイヤ layer_index から layer_size 個のレイヤの，座標 (col_index，row_index)，サイズ (col_size，row_size) の矩形領域を dest_img が示すオブジェクトにコピーします.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤをコピーの対象とします.

image().cut() の場合は，dest_img が示すオブジェクトにコピーした後，該当領域は値 0 で埋められます.

**PARAMETER**

| | | |
|---|---|---|
| [O] | dest_img | 指定された矩形領域の保存先となるオブジェクト |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

cut() メンバ関数は当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部メモリ領域確保の操作に失敗した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは (0,0) から 128 × 128 の正方形の領域を my_image_buffer にコピーし，元イメージの (128,0) にそれを貼り付けます.

```
fits_image my_image_buffer;
fits.image("Primary").copy(&my_image_buffer, 0,128, 0,128);
fits.image("Primary").paste(my_image_buffer, 128, 0);
```

チュートリアルの §5.10 にも使用例があります.

---

### 12.6.35 image().paste()

**NAME**

image().paste() — コピーバッファのイメージを貼り付け

**SYNOPSIS**

```
fits_image &image( ... ).paste( const fits_image &src_img,
                                long col_index = 0, long row_index = 0,
                                long layer_index = 0 );
```

**DESCRIPTION**

layer_index で指定されたレイヤの，座標 (col_index，row_index) へ，src_img で示されたオブジェクトの内容を貼り付けます.

自身の型と src_img の型が一致していない場合や，ヘッダの BZERO，BSCALE，BLANK が異なる場合も，適切に変換を行ないます.

**PARAMETER**

| | | |
|---|---|---|
| [I] | src_img | コピー元となるオブジェクト |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

内部メモリ領域確保の操作に失敗した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.6.34の EXAMPLES を参照してください．チュートリアルの §5.10 にも使用例があります.

---

### 12.6.36 image().add()

**NAME**

image().add() — コピーバッファのイメージを元のイメージに加算

**SYNOPSIS**

```
fits_image &image( ... ).add( const fits_image &src_img,
                              long col_index = 0, long row_index = 0,
                              long layer_index = 0 );
```

**DESCRIPTION**

layer_index で指定されたレイヤの，座標 (col_index，row_index) へ，src_img で示され

たオブジェクト内のイメージを元のイメージに加算します．src_img のピクセルが NAN の場合は，当該ピクセルは変更されません．

自身の型と src_img の型が一致していない場合や，ヘッダの BZERO，BSCALE，BLANK が異なる場合も，適切に変換を行ないます．

**PARAMETER**

[I] src_img 源泉となるオブジェクト
[I] col_index カラム ($x$ 軸) ピクセル位置
[I] row_index ロウ ($y$ 軸) ピクセル位置
[I] layer_index レイヤ ($z$ 軸) ピクセル位置
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXAMPLES**

§12.6.34の EXAMPLES の paste() の部分を以下のように書き換えると，元の画像と重ねたものができます．

```
fits.image("Primary").add(my_image_buffer, 128, 0);
```

---

### 12.6.37 image().subtract()

**NAME**

image().subtract() — 元のイメージからコピーバッファのイメージを減算

**SYNOPSIS**

```
fits_image &image( ... ).subtract( const fits_image &src_img,
                                   long col_index = 0, long row_index = 0,
                                   long layer_index = 0 );
```

**DESCRIPTION**

layer_index で指定されたレイヤの，座標 (col_index，row_index) で，元のイメージから src_img で示されたオブジェクト内のイメージを減算します．src_img のピクセルが NAN の場合は，当該ピクセルは変更されません．

自身の型と src_img の型が一致していない場合や，ヘッダの BZERO，BSCALE，BLANK が異なる場合も，適切に変換を行ないます．

**PARAMETER**

[I] src_img 源泉となるオブジェクト
[I] col_index カラム ($x$ 軸) ピクセル位置
[I] row_index ロウ ($y$ 軸) ピクセル位置
[I] layer_index レイヤ ($z$ 軸) ピクセル位置
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXAMPLES**

§12.6.36の EXAMPLES を参照してください.

---

### 12.6.38 image().multiply()

**NAME**

image().multiply() — 元のイメージにコピーバッファのイメージを乗算

**SYNOPSIS**

```
fits_image &image( ... ).multiply( const fits_image &src_img,
                                   long col_index = 0, long row_index = 0,
                                   long layer_index = 0 );
```

**DESCRIPTION**

layer_index で指定されたレイヤの，座標 (col_index，row_index) で，元のイメージに src_img で示されたオブジェクト内のイメージを乗算します．src_img のピクセルが NAN の場合は，当該ピクセルは変更されません.

自身の型と src_img の型が一致していない場合や，ヘッダの BZERO，BSCALE，BLANK が異なる場合も，適切に変換を行ないます.

**PARAMETER**

| | | |
|---|---|---|
| [I] | src_img | 源泉となるオブジェクト |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES**

§12.6.36の EXAMPLES を参照してください.

---

### 12.6.39 image().divide()

**NAME**

image().divide() — 元のイメージをコピーバッファのイメージで除算

**SYNOPSIS**

```
fits_image &image( ... ).divide( const fits_image &src_img,
                                 long col_index = 0, long row_index = 0,
                                 long layer_index = 0 );
```

**DESCRIPTION**

layer_index で指定されたレイヤの，座標 (col_index，row_index) で，元のイメージを src_img で示されたオブジェクト内のイメージで除算します．src_img のピクセルが NAN の場合は，当該ピクセルは変更されません.

自身の型と src_img の型が一致していない場合や，ヘッダの BZERO，BSCALE，BLANK が異なる場合も，適切に変換を行ないます.

**PARAMETER**

[I] src_img 源泉となるオブジェクト
[I] col_index カラム ($x$ 軸) ピクセル位置
[I] row_index ロウ ($y$ 軸) ピクセル位置
[I] layer_index レイヤ ($z$ 軸) ピクセル位置
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES**

§12.6.36の EXAMPLES を参照してください.

---

### 12.6.40 image().paste()

**NAME**

image().paste() — コピーバッファのイメージを貼り付け (ユーザ関数付き)

**SYNOPSIS**

```
fits_image &image( ... ).paste( const fits_image &src_img,
    void (*func)(double [],double [],long, long,long,long,fits_image *,void *),
    void *user_ptr,
    long dest_col = 0, long dest_row = 0, long dest_layer = 0 );
```

**DESCRIPTION**

dest_layer で指定されたレイヤの，座標 (dest_col，dest_row) に，src_img で示されたオブジェクト内のイメージを貼り付けます．この時，ユーザ関数 (*func) でピクセル値を変更する事ができます.

このメンバ関数では，ユーザ関数でのデータ入出力用のテンポラリバッファが用意され，指定領域内で 1 行ごとのラスタースキャン (横方向行単位でのスキャン) が行なわれて，それぞれの行で，(1) テンポラリバッファにオブジェクトのピクセルバッファの内容を変換・コピー，(2) ユーザ関数の呼び出し，(3) テンポラリバッファの内容をオブジェクトのピクセルバッファへ変換・コピー，が行なわれます．この時，SCALE，ZERO，BLANK の変換も適切に行なわれます．なお，ユーザ関数の中では，横 1 列分のピクセルを変更するためのループを組む必要があります.

ユーザ関数 (*func) の引数の仕様は次のとおりで，変更結果を pix_self[] に対して書き込むようにコードを組んでください.

double pix_self[] … 元のピクセル値
double pix_src[] … src_img のピクセル値
long n_pix ... pix_self または pix_src の個数
long axis0 … $x$ 軸の座標
long axis1 … $y$ 軸の座標
long axis2 … $z$ 軸の座標

`fits_image *thisp` … 自身のオブジェクトのアドレス
`void *ptr` … `user_ptr` の値

`user_ptr` はユーザが自由に使えるポインタ変数で，この値はユーザ関数の最後の引数に与えられます.

カラム，ロウ，レイヤの指定を省略した場合，それぞれすべてのカラム，すべてのロウ，すべてのレイヤを対象とします．ただし，ユーザは定数 `FITS::ALL` を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `src_img` | コピー元となるオブジェクト |
| [I] | `func` | ユーザ関数のアドレス |
| [I] | `user_ptr` | ユーザ関数に与えられる任意のポインタ変数 |
| [I] | `dest_col` | コピー先のカラム ($x$ 軸) ピクセル位置 |
| [I] | `dest_row` | コピー先のロウ ($y$ 軸) ピクセル位置 |
| [I] | `dest_layer` | コピー先のレイヤ ($z$ 軸) ピクセル位置 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXAMPLES**

ユーザ関数の指定例については，§12.6.33の EXAMPLES を参照してください.

---

### 12.6.41 image().stat_pixels()

**NAME**

image().stat_pixels() — 指定領域の平均値，標準偏差，中央値などの算出

**SYNOPSIS**

```
long image( ... ).stat_pixels( double results[], size_t results_len,
                      const char *options,
                      long col_index = 0, long col_size = FITS::ALL,
                      long row_index = 0, long row_size = FITS::ALL,
                      long layer_index = 0, long layer_size = FITS::ALL ) const;
long image( ... ).stat_pixels( fits_header *results, const char *options,
                      long col_index = 0, long col_size = FITS::ALL,
                      long row_index = 0, long row_size = FITS::ALL,
                      long layer_index = 0, long layer_size = FITS::ALL ) const;
```

**DESCRIPTION**

`col_index` 以降の引数によって指定された領域のピクセル値の統計値 (平均値，標準偏差，中央値など) を計算し，`results` で示された配列またはオブジェクトにその結果を格納します.

どの統計値が必要かは引数 `options` に指定します．例えば，`"results=mean,stddev,median"` とすると，平均値，標準偏差，中央値を計算します.「`results=`」の後に指定できる文字列は次のとおり:

`npix` ... 総ピクセル数

mean ... 平均値
stddev ... 標準偏差
median ... 中央値[38)]
min ... 最小値
max ... 最大値
skew ... 歪度
kurtosis ... 尖度

SYNOPSIS において，前者の場合は results の配列に options で指定された順に結果が入り，後者の場合は results の FITS ヘッダオブジェクトにキーワード (例えば"MEAN") と値のセットが格納されます.

**PARAMETER**

| | | |
|---|---|---|
| [O] | results | 結果を格納するための配列またはオブジェクトのアドレス |
| [I] | results_len | 配列 results の長さ |
| [I] | options | 統計値の計算に関するオプション文字列 |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : 有効ピクセルの数.
負の値 (エラー) : 引数の座標値が不正等の場合.

**EXAMPLES**

次のコードは，Primary HDU の画像の統計値を求め，表示します．fits_header クラスのオブジェクトを統計値の結果の格納のために使う事ができます.

```
fits_header stat_results;
const char *stat_options = "results=npix,mean,stddev,min,max,median";
long i;
/* 統計値を計算 */
if ( fits.image("Primary").stat_pixels(&stat_results, stat_options) < 0 ) {
    /* エラー処理 */
}
/* 結果を表示 */
for ( i=0 ; i < stat_results.length() ; i++ ) {
    printf("%s = %.15g\n", stat_results.at(i).keyword(),
           stat_results.at(i).dvalue());
}
```

[38)] これは IRAF の midpt(median の近似値) ではありません．SFITSIO では正真正銘の median が高速に計算されます.

結果を取り出す場合に，stat_results.at("NPIX").dvalue() のように連想配列として扱う事もできます.

§1.1の課題 3 の解答例，SFITSIO のパッケージに含まれる tools/stat_pixels.cc もあわせてご覧ください.

---

### 12.6.42 image().combine_layers()

**NAME**

image().combine_layers() — 三次元画像の指定領域を，平均値や中央値等で画像コンバイン

**SYNOPSIS**

```
long image( ... ).combine_layers( fits_image *dest_img, const char *options,
                      long col_index = 0, long col_size = FITS::ALL,
                      long row_index = 0, long row_size = FITS::ALL,
                      long layer_index = 0, long layer_size = FITS::ALL ) const;
```

**DESCRIPTION**

col_index 以降の引数によって指定された領域について，それぞれの $(x, y)$ のピクセルについて $z$ 方向にピクセル値をコンバインして二次元の画像を作成し，引数 dest_img が示すオブジェクトに格納します.

どのようにコンバインし，結果を格納するかは引数 options で指定します．例えば，

"combine=average outtype=float"

とすると，コンバインは平均値で行なわれ，結果は 4 バイト浮動小数点値で出力されます.

「combine=」の後には，コンバインに用いる手法を指定し，次の文字列が使えます:

average ... 平均値
median ... 中央値
sum ... 合計値
min ... 最小値
max ... 最大値

「outtype=」の後には，出力する画像のタイプを指定し，次の文字列が使えます:

short ... 2 バイト符号付き整数
ushort ... 2 バイト符号無し整数 (BZERO=32768.0)
long ... 4 バイト符号付き整数
longlong ... 8 バイト符号付き整数
float ... 4 バイト浮動小数点数
double ... 8 バイト浮動小数点数

**PARAMETER**

| | | |
|---|---|---|
| [O] | dest_img | 結果を格納するためのオブジェクトのアドレス |
| [I] | options | コンバインに関するオプション文字列 |
| [I] | col_index | カラム ($x$ 軸) ピクセル位置 |
| [I] | col_size | col_index からのピクセル数 |
| [I] | row_index | ロウ ($y$ 軸) ピクセル位置 |
| [I] | row_size | row_index からのピクセル数 |
| [I] | layer_index | レイヤ ($z$ 軸) ピクセル位置 |
| [I] | layer_size | layer_index からのピクセル数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : 有効ピクセルの数.
負の値 (エラー) : 引数の座標値が不正等の場合.

**EXAMPLES**

次のコードは，3 枚の二次元画像を中央値でコンバインする例です．最初の fitscc オブジェクト「fits0」に最初の FITS ファイルを読み込み，3 次元化して 2 枚目と 3 枚目の FITS ファイルの画像を奥のレイヤに貼り付け，最後にコンバインします．なお，可読性を高めるため，エラー処理は省略しています.

```
fitscc fits0, fits1;
/* read and prepare */
fits0.read_stream("raw_image_0.fits");      /* 2d image No.1 */
fits0.image("Primary").resize(2, 3);        /* resize z-axis => 3 */
fits1.read_stream("raw_image_1.fits");      /* 2d image No.2 */
fits0.image("Primary").paste(fits1.image("Primary"), 0L, 0L, 1L);
fits1.read_stream("raw_image_2.fits");      /* 2d image No.3 */
fits0.image("Primary").paste(fits1.image("Primary"), 0L, 0L, 2L);
/* combine (fits1.image("Primary") will be overwritten) */
fits0.image("Primary").combine_layers(&fits1.image("Primary"),
                                      "combine=median outtype=float");
/* output combined image */
fits1.write_stream("combined_image.fits");
```

SFITSIO のパッケージに含まれる tools/combine_images.cc もご覧ください.

---

## 12.7 Image HDU の操作 (低レベル)

ここでは，Image HDU を操作するための低レベル API を解説します．低レベル API では，ヘッダの BZERO，BSCALE の値による変換処理ついてはユーザ自身で行なう必要があります．通常，ここで取り上げる API は必要ありませんが，高速化などのためには有用かもしれません.

「image( ... )」の括弧内の引数はすべて同じですので，その部分の引数に関する記述は省略します．この括弧内の引数には，HDU 番号 (long index) あるいは HDU 名 (const char *hduname) を指定します.

### 12.7.1 image().data_array()

**NAME**

image().data_array() — データバッファ管理オブジェクトの参照

**SYNOPSIS**

```
sli::mdarray &image( ... ).data_array();
const sli::mdarray &image( ... ).data_array() const;
const sli::mdarray &image( ... ).data_array_cs() const;
```

**DESCRIPTION**

fits_image クラスのイメージバッファは，SLLIB の mdarray クラスを使って管理しています．data_array() メンバ関数は，ユーザが mdarray クラスを使って演算等を行ないたい場合に利用します．

mdarray クラスの詳細は，SLLIB のマニュアルを参照してください．

---

### 12.7.2 image().data_ptr()

**NAME**

image().data_ptr() — オブジェクト内データバッファのアドレス

**SYNOPSIS**

```
void *image( ... ).data_ptr( long axis0 = 0,
                             long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
void *image( ... ).data_ptr_v( long num_axisx,
                               long axis0, long axis1, long axis2, ... );
void *image( ... ).va_data_ptr_v( long num_axisx,
                                  long axis0, long axis1, long axis2, va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の，内部イメージデータバッファのアドレスを返します．引数の座標値が範囲外の場合，NULL を返します．

axis0，axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます．

返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，バッファの型やサイズが変更された場合は無効になります．

返されたアドレスは，現在の FITS の型に応じて，fits::double_t *，fits::float_t *，fits::longlong_t *，fits::long_t *，fits::short_t *，fits::byte_t * のいずれかの型にキャストして使います．

ユーザは定数 FITS::INDEF を明示的に使わないでください．

**PARAMETER**

| | | |
|---|---|---|
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

| | | |
|---|---|---|
| 整数値 | ： | 内部イメージデータバッファのアドレス. |
| NULL(エラー) | ： | 引数の座標値が範囲外の場合. |

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits::double_t *img_ptr = (fits::double_t *)fits.image("Primary").data_ptr();
```

チュートリアルの §5.11 にも使用例があります.

---

### 12.7.3 image().get_data()

**NAME**

image().get_data() — 生のイメージデータを取得

**SYNOPSIS**

```
ssize_t image( ... ).get_data( void *dest_buf, size_t buf_size,
                               long axis0 = 0, long axis1 = FITS::INDEF,
                               long axis2 = FITS::INDEF ) const;
ssize_t image( ... ).get_data_v( void *dest_buf, size_t buf_size,
                                 long num_axisx,
                                 long axis0, long axis1, long axis2, ... ) const;
ssize_t image( ... ).va_get_data_v( void *dest_buf, size_t buf_size,
                                    long num_axisx,
                                    long axis0, long axis1, long axis2,
                                    va_list ap ) const;
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の，生のイメージデータを，最大で buf_size バイト，dest_buf へコピーします.

axis0，axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

dest_buf で指定したアドレスは，現在の FITS の型に応じて，fits::double_t *，fits::float_t *，fits::longlong_t *，fits::long_t *，fits::short_t *，fits::byte_t * のいずれかの型にキャストして使います.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [O] | dest_buf | イメージデータの取得領域アドレス |
| [I] | buf_size | dest_buf のバイトサイズ |
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 ： バッファ長が十分な場合にコピーできるバイト数.
負の値 (エラー) ： 引数が不正でコピーされなかった場合.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
size_t buf_size = fits.image("Primary").bytes() * fits.image("Primary").length();
fits::double_t *dest_buf = (fits::double_t *)malloc(buf_size);
if ( dest_buf == NULL ) {
    エラー処理
}
fits.image("Primary").get_data(dest_buf, buf_size);
```

---

### 12.7.4 image().put_data()

**NAME**

image().put_data() — 生のイメージデータをセット

**SYNOPSIS**

```
ssize_t image( ... ).put_data( const void *src_buf, size_t buf_size, long axis0 = 0,
                               long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
ssize_t image( ... ).put_data_v( const void *src_buf, size_t buf_size,
                                 long num_axisx,
                                 long axis0, long axis1, long axis2, ... );
ssize_t image( ... ).va_put_data_v( const void *src_buf, size_t buf_size,
                                    long num_axisx,
                                    long axis0, long axis1, long axis2,
                                    va_list ap );
```

**DESCRIPTION**

src_buf の生イメージデータを，オブジェクトのカラム axis0，ロウ axis1，レイヤ axis2 へ，最大で buf_size バイトコピーします.

axis0, axis1, axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | src_buf | コピー元となるイメージデータ領域のアドレス |
| [I] | buf_size | src_buf のバイトサイズ |
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : バッファ長が十分な場合にコピーできるバイト数.
負の値 (エラー) : 引数が不正でコピーされなかった場合.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
size_t buf_size = fits.image("Primary").bytes() * fits.image("Primary").length();
fits::double_t *src_buf = (fits::double_t *)malloc(buf_size);
                :
                :
fits.image("Primary").put_data(src_buf, buf_size);
```

---

### 12.7.5 image().double_value()

**NAME**

image().double_value() — 生のイメージデータを返す

**SYNOPSIS**

```
double image( ... ).double_value( long axis0, long axis1 = FITS::INDEF,
                                  long axis2 = FITS::INDEF ) const;
double image( ... ).double_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, ... ) const;
double image( ... ).va_double_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, va_list ap ) const;
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値を返します．ヘッダの BZERO, BSCALE を反映した値が必要な場合は，image().dvalue()(§12.6.10) を使います.

axis1, axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードは，0 行目の全てのピクセル値を表示しています.

```
long idx;
for ( idx=0 ; idx < fits.image("Primary").col_length() ; idx++ ) {
    printf("index[%ld]=[%lf]\n", idx, fits.image("Primary").double_value(idx,0));
}
```

---

### 12.7.6 image().float_value()

**NAME**

image().float_value() — 生のイメージデータを返す

**SYNOPSIS**

```
float image( ... ).float_value( long axis0, long axis1 = FITS::INDEF,
                                long axis2 = FITS::INDEF ) const;
float image( ... ).float_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, ... ) const;
float image( ... ).va_float_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, va_list ap ) const;
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値を返します．ヘッダの BZERO, BSCALE を反映した値が必要な場合は，image().dvalue()(§12.6.10) を使います.

axis1, axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

[I] `axis0` カラム ($x$ 軸) ピクセル位置の指定
[I] `axis1` ロウ ($y$ 軸) ピクセル位置の指定
[I] `axis2` レイヤ ($z$ 軸) ピクセル位置の指定
[I] `num_axisx` 引数で指定している次元軸の数
[I] `...` 各次元内のピクセル位置全要素データ
[I] `ap` 各次元内のピクセル位置全要素データ

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

§12.7.5の EXAMPLES を参照してください.

---

### 12.7.7 image().longlong_value()

**NAME**

image().longlong_value() — 生のイメージデータを返す

**SYNOPSIS**

```
long long image( ... ).longlong_value( long axis0, long axis1 = FITS::INDEF,
                                       long axis2 = FITS::INDEF ) const;
long long image( ... ).longlong_value_v( long num_axisx,
                         long axis0, long axis1, long axis2, ... ) const;
long long image( ... ).va_longlong_value_v( long num_axisx,
                         long axis0, long axis1, long axis2, va_list ap ) const;
```

**DESCRIPTION**

カラム `axis0`，ロウ `axis1`，レイヤ `axis2` の生のピクセル値を返します．ヘッダの `BZERO`，`BSCALE` を反映した値が必要な場合は，`image().dvalue()`，`image().lvalue()`，`image().llvalue()` のいずれかを使います (§12.6.10〜).

`axis1`，`axis2` は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 `FITS::INDEF` を明示的に使わないでください.

**PARAMETER**

[I] `axis0` カラム ($x$ 軸) ピクセル位置の指定
[I] `axis1` ロウ ($y$ 軸) ピクセル位置の指定
[I] `axis2` レイヤ ($z$ 軸) ピクセル位置の指定
[I] `num_axisx` 引数で指定している次元軸の数
[I] `...` 各次元内のピクセル位置全要素データ
[I] `ap` 各次元内のピクセル位置全要素データ

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

§12.7.5の EXAMPLES を参照してください.

---

### 12.7.8 image().long_value()

**NAME**

image().long_value() — 生のイメージデータを返す

**SYNOPSIS**

```
long image( ... ).long_value( long axis0, long axis1 = FITS::INDEF,
                              long axis2 = FITS::INDEF ) const;
long image( ... ).long_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, ... ) const;
long image( ... ).va_long_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, va_list ap ) const;
```

**DESCRIPTION**

カラム `axis0`，ロウ `axis1`，レイヤ `axis2` の生のピクセル値を返します．ヘッダの `BZERO`, `BSCALE` を反映した値が必要な場合は，`image().dvalue()`, `image().lvalue()`, `image().llvalue()` のいずれかを使います (§12.6.10〜).

`axis1`，`axis2` は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 `FITS::INDEF` を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `axis0` | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | `axis1` | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | `axis2` | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | `num_axisx` | 引数で指定している次元軸の数 |
| [I] | `...` | 各次元内のピクセル位置全要素データ |
| [I] | `ap` | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

§12.7.5の EXAMPLES を参照してください.

---

### 12.7.9 image().short_value()

**NAME**

image().short_value() — 生のイメージデータを返す

**SYNOPSIS**

```
short image( ... ).short_value( long axis0, long axis1 = FITS::INDEF,
                                long axis2 = FITS::INDEF ) const;
short image( ... ).short_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, ... ) const;
short image( ... ).va_short_value_v( long num_axisx,
                        long axis0, long axis1, long axis2, va_list ap ) const;
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値を返します．ヘッダの BZERO，BSCALE を反映した値が必要な場合は，image().dvalue()，image().lvalue()，image().llvalue() のいずれかを使います (§12.6.10〜).

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.7.5の EXAMPLES を参照してください.

---

### 12.7.10 image().byte_value()

**NAME**

image().byte_value() — 生のイメージデータを返す

**SYNOPSIS**

```
unsigned char image( ... ).byte_value( long axis0, long axis1 = FITS::INDEF,
                                       long axis2 = FITS::INDEF ) const;
unsigned char image( ... ).byte_value_v( long num_axisx,
                         long axis0, long axis1, long axis2, ... ) const;
unsigned char image( ... ).va_byte_value_v( long num_axisx,
                         long axis0, long axis1, long axis2, va_list ap ) const;
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値を返します．ヘッダの BZERO，BSCALE を反映した値が必要な場合は，image().dvalue()，image().lvalue()，image().llvalue() のいずれかを使います (§12.6.10〜)．

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます．

ユーザは定数 FITS::INDEF を明示的に使わないでください．

**PARAMETER**

| | | |
|---|---|---|
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

ピクセル値を返します．

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

§12.7.5の EXAMPLES を参照してください．

---

### 12.7.11 image().assign_double()

**NAME**

image().assign_double() — 生の値をセット

**SYNOPSIS**

```
fits_image &image( ... ).assign_double( double value, long axis0,
```

```
                                long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
fits_image &image( ... ).assign_double_v( double value, long num_axisx,
                                        long axis0, long axis1, long axis2, ... );
fits_image &image( ... ).va_assign_double_v( double value, long num_axisx,
                                    long axis0, long axis1, long axis2, va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値をセットします．値 value が，ヘッダの BZERO，BSCALE の値による変換を必要とする場合は，image().assign()(§12.6.12) を使います．

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます．

ユーザは定数 FITS::INDEF を明示的に使わないでください．

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | セットするピクセル値 |
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

次のコードは，0 行目のすべてのピクセルを値 0 に変更しています．

```
long idx;
for ( idx=0 ; idx < fits.image("Primary").col_length() ; idx++ ) {
    fits.image("Primary").assign_double(0.0, idx,0);
}
```

---

### 12.7.12 image().assign_float()

**NAME**

image().assign_float() — 生の値をセット

**SYNOPSIS**

```
fits_image &image( ... ).assign_float( float value, long axis0,
```

```
                              long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
fits_image &image( ... ).assign_float_v( float value, long num_axisx,
                                         long axis0, long axis1, long axis2, ... );
fits_image &image( ... ).va_assign_float_v( float value, long num_axisx,
                                    long axis0, long axis1, long axis2, va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値をセットします．値 value が，ヘッダの BZERO，BSCALE の値による変換を必要とする場合は，image().assign()(§12.6.12) を使います．

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます．

ユーザは定数 FITS::INDEF を明示的に使わないでください．

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | セットするピクセル値 |
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

§12.7.11の EXAMPLES を参照してください．

---

### 12.7.13 image().assign_longlong()

**NAME**

image().assign_longlong() — 生の値をセット

**SYNOPSIS**

```
fits_image &image( ... ).assign_longlong( long long value, long axis0,
                              long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
fits_image &image( ... ).assign_longlong_v( long long value, long num_axisx,
                                         long axis0, long axis1, long axis2, ... );
fits_image &image( ... ).va_assign_longlong_v( long long value, long num_axisx,
                                    long axis0, long axis1, long axis2, va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値をセットします．値 value が，ヘッダの BZERO，BSCALE の値による変換を必要とする場合は，image().assign()(§12.6.12) を使います．

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます．

ユーザは定数 FITS::INDEF を明示的に使わないでください．

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | セットするピクセル値 |
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します．

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

§12.7.11の EXAMPLES を参照してください．

---

### 12.7.14 image().assign_long()

**NAME**

image().assign_long() — 生の値をセット

**SYNOPSIS**

```
fits_image &image( ... ).assign_long( long value, long axis0,
                            long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
fits_image &image( ... ).assign_long_v( long value, long num_axisx,
                                        long axis0, long axis1, long axis2, ... );
fits_image &image( ... ).va_assign_long_v( long value, long num_axisx,
                                    long axis0, long axis1, long axis2, va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値をセットします．値 value が，ヘッダの BZERO，BSCALE の値による変換を必要とする場合は，image().assign()(§12.6.12) を使います．

axis1, axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | セットするピクセル値 |
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.7.11の EXAMPLES を参照してください.

---

### 12.7.15 image().assign_short()

**NAME**

image().assign_short() — 生の値をセット

**SYNOPSIS**

```
fits_image &image( ... ).assign_short( short value, long axis0,
                           long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
fits_image &image( ... ).assign_short_v( short value, long num_axisx,
                                         long axis0, long axis1, long axis2, ... );
fits_image &image( ... ).va_assign_short_v( short value, long num_axisx,
                                    long axis0, long axis1, long axis2, va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値をセットします．値 value が，ヘッダの BZERO，BSCALE の値による変換を必要とする場合は，image().assign()(§12.6.12) を使います.

axis1, axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | セットするピクセル値 |
| [I] | axis0 | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | axis1 | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | axis2 | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | num_axisx | 引数で指定している次元軸の数 |
| [I] | ... | 各次元内のピクセル位置全要素データ |
| [I] | ap | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.7.11の EXAMPLES を参照してください.

---

### 12.7.16 image().assign_byte()

**NAME**

image().assign_byte() — 生の値をセット

**SYNOPSIS**

```
fits_image &image( ... ).assign_byte( unsigned char value, long axis0,
                          long axis1 = FITS::INDEF, long axis2 = FITS::INDEF );
fits_image &image( ... ).assign_byte_v( unsigned char value, long num_axisx,
                                    long axis0, long axis1, long axis2, ... );
fits_image &image( ... ).va_assign_byte_v( unsigned char value, long num_axisx,
                                long axis0, long axis1, long axis2, va_list ap );
```

**DESCRIPTION**

カラム axis0，ロウ axis1，レイヤ axis2 の生のピクセル値をセットします．値 value が，ヘッダの BZERO，BSCALE の値による変換を必要とする場合は，image().assign()(§12.6.12) を使います.

axis1，axis2 は省略可能で，引数の個数によって $n$ 次元のデータを，1 次元，2 次元あるいは 3 次元 (以上) として扱う事ができます.

ユーザは定数 FITS::INDEF を明示的に使わないでください.

**PARAMETER**

| | | |
|---|---|---|
| [I] | `value` | セットするピクセル値 |
| [I] | `axis0` | カラム ($x$ 軸) ピクセル位置の指定 |
| [I] | `axis1` | ロウ ($y$ 軸) ピクセル位置の指定 |
| [I] | `axis2` | レイヤ ($z$ 軸) ピクセル位置の指定 |
| [I] | `num_axisx` | 引数で指定している次元軸の数 |
| [I] | `...` | 各次元内のピクセル位置全要素データ |
| [I] | `ap` | 各次元内のピクセル位置全要素データ |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_image オブジェクトの参照を返します.

**EXCEPTION**

可変引数に不整合な値を指定した場合，SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

§12.7.11の EXAMPLES を参照してください.

---

## 12.8 Ascii Table HDU・Binary Table HDU の操作

ここでは，Ascii Table HDU と Binary Table HDU を操作するための API を解説します．SFITSIO では，アスキーテーブルはバイナリテーブルの文字列カラム (ヘッダの TTYPE$n$ が $x$A の場合) だけの場合として扱うので，アスキーテーブルもバイナリテーブルも全く同じメンバ関数で扱う事ができます．

API は 2 種類に大別されます．1 つめのケースは，

```
value = fits.table( ... ).function( ... );
```

の形，2 つめのケースは

```
value = fits.table( ... ).col( ... ).function( ... );
value = fits.table( ... ).colf( ... ).function( ... );
```

の形です．

table( ... ) の括弧内の引数には，HDU 番号 (long index) あるいは HDU 名 (const char *hduname) を指定します．

col() の括弧内の引数には，カラムのインデックス (long index) あるいはカラム名 (const char *col_name) を指定します．colf() の括弧内の引数には，カラム名を libc の printf() 関数と同様に指定します．

これ以降，「table( ... )」「col( ... )」「colf( ... )」の括弧内の引数はすべて同じですので，その部分の引数に関する記述は省略します．

### 12.8.1 table().hduname(), table().assign_hduname()

**NAME**

table().assign_hduname(), table().hduname() — HDU 名の操作

**SYNOPSIS**

```
const char *table( ... ).hduname();
const char *table( ... ).extname();
fits_table &table( ... ).assign_hduname( const char *name );
fits_table &table( ... ).assign_extname( const char *name );
```

**DESCRIPTION**

table().hduname() メンバ関数は，HDU の名称を返します．

table().assign_hduname() メンバ関数は，HDU の名称を変更します．名称 name はヘッダ EXTNAME に反映されます．

**PARAMETER**

[I] name セットする HDU 名称

([I]：入力，[O]：出力)

**RETURN VALUE**

table().hduname() は HDU 名称文字列のアドレスを返します．

table().assign_hduname() は当該 fits_table オブジェクトへの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，HDU 名称変更時の領域リサイズ処理で失敗した場合)，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
printf("HDU 名=%s\n", fits.table("EVENT").hduname());
```

---

### 12.8.2 table().hduver(), table().assign_hduver()

**NAME**

table().assign_hduver(), table().hduver() — HDU のバージョン番号の操作

**SYNOPSIS**

```
long long table( ... ).hduver();
long long table( ... ).extver();
fits_table &table( ... ).assign_hduver( long long ver );
fits_table &table( ... ).assign_extver( long long ver );
```

**DESCRIPTION**

`table().hduver()` メンバ関数は，HDU のバージョン番号を返します.

`table().assign_hduver()` メンバ関数は，HDU のバージョン番号を変更します．名称 `ver` はヘッダ `EXTVER` に反映されます.

**PARAMETER**

[I] `ver` セットするバージョン番号
([I]：入力，[O]：出力)

**RETURN VALUE**

`table().hduver()` は HDU のバージョン番号を返します.
`table().assign_hduver()` は当該 fits_table オブジェクトへの参照を返します.

**EXAMPLES**

```
printf("HDU バージョン=%lld\n", fits.table("EVENT").hduver());
```

---

### 12.8.3 table().col_length()

**NAME**

table().col_length() — カラムの数

**SYNOPSIS**

```
long table( ... ).col_length() const;
```

**RETURN VALUE**

テーブルのカラム数を返します.

**EXAMPLES**

```
printf("カラム数=%ld\n", fits.table("EVENT").col_length());
```

---

### 12.8.4 table().row_length()

**NAME**

table().row_length() — 行の数

**SYNOPSIS**

```
long table( ... ).row_length() const;
```

**RETURN VALUE**

テーブルの行数を返します.

**EXAMPLES**

```
printf("行数=%ld\n", fits.table("EVENT").row_length());
```

---

### 12.8.5 table().heap_length()

**NAME**

table().heap_length() — ヒープ領域のバイト長

**SYNOPSIS**

```
long table( ... ).heap_length() const;
```

**RETURN VALUE**

テーブルに属しているヒープ領域のバイト長を返します.

---

### 12.8.6 table().col_index()

**NAME**

table().col_index() — カラムの番号

**SYNOPSIS**

```
long table( ... ).col_index( const char *col_name ) const;
```

**DESCRIPTION**

`table().col\_index()` メンバ関数は，`col_name` で指定されたカラムの番号を返します.

**PARAMETER**

[I] `col_name` カラム名
([I] : 入力，[O] : 出力)

**RETURN VALUE**

非負の値 : カラム番号.
負の値 (エラー) : 見つからなかった場合.

**EXAMPLES**

次のコードは，テーブル内の全てのカラム名とインデックス番号を表示します.

```
long idx;
for ( idx=0 ; idx < fits.table("EVENT").col_length() ; idx++ ) {
    const char *col_name = fits.table("EVENT").col_name(idx);
    long col_idx = fits.table("EVENT").col_index(col_name);
    printf("カラム名=%s\t カラム番号=%ld\n", col_name, col_idx);
}
```

---

### 12.8.7 table().col_name()

**NAME**

table().col_name() — カラムの名前

**SYNOPSIS**

```
const char *table( ... ).col_name( long col_index ) const;
```

**DESCRIPTION**

`col_index` で指定されたカラムの名前を返します.

**PARAMETER**

[I] `col_index` カラムインデックス

([I]：入力，[O]：出力)

**RETURN VALUE**

カラムの名前文字列のアドレスを返します.

**EXAMPLES**

§12.8.6の EXAMPLES を参照してください.

---

### 12.8.8 table().col().type()

**NAME**

table().col().type() — カラムの型

**SYNOPSIS**

```
int table( ... ).col( ... ).type() const;
```

**DESCRIPTION**

`table().col().type()` メンバ関数は，カラムの型を返します.

値は, `FITS::DOUBLE_T`, `FITS::FLOAT_T`, `FITS::LONGLONG_T`, `FITS::LONG_T`, `FITS::SHORT_T`, `FITS::BYTE_T`, `FITS::BIT_T`, `FITS::LOGICAL_T`, `FITS::COMPLEX_T`, `FITS::DOUBLECOMPLEX_T`, `FITS::ASCII_T`, `FITS::LONGARRDESC_T`, `FITS::LLONGARRDESC_T` のいずれかで返されます.

**EXAMPLES**

次のコードは，テーブル内の全てのカラムについて次の情報を取得します.

・カラムの型
・カラム型のバイト数
・カラムの 1 行の要素数
・カラムの 1 行のバイト数

```
long idx;
for ( idx=0 ; idx < fits.table("EVENT").col_length() ; idx++ ) {
    int c_type = fits.table("EVENT").col(idx).type();
    long c_bytes = fits.table("EVENT").col(idx).bytes();
    long c_elem_len = fits.table("EVENT").col(idx).elem_length();
    long c_elem_byte_len = fits.table("EVENT").col(idx).elem_byte_length();
                :
                :
}
```

---

### 12.8.9 table().col().heap_is_used()

**NAME**

table().col().heap_is_used() — 可変長配列かどうかを判定

**SYNOPSIS**

```
bool table( ... ).col( ... ).heap_is_used() const;
```

**DESCRIPTION**

`table().col().heap_is_used()` メンバ関数は，当該カラムが可変長配列である場合 (ヒープを使っている場合) は `true` を，そうでない場合は `false` を返します.

---

### 12.8.10 table().col().heap_type()

**NAME**

table().col().heap_type() — カラムのヒープ (可変長配列) のデータ型

**SYNOPSIS**

```
int table( ... ).col( ... ).heap_type() const;
```

**DESCRIPTION**

`table().col().heap_type()` メンバ関数は，カラムで使われているヒープ (可変長配列) の型を返します.

値は, `FITS::DOUBLE_T`, `FITS::FLOAT_T`, `FITS::LONGLONG_T`, `FITS::LONG_T`, `FITS::SHORT_T`, `FITS::BYTE_T`, `FITS::BIT_T`, `FITS::LOGICAL_T`, `FITS::COMPLEX_T`, `FITS::DOUBLECOMPLEX_T`, `FITS::ASCII_T` のいずれかで返されます.

---

### 12.8.11 table().col().bytes()

**NAME**

table().col().bytes() — カラムの型のバイト数

**SYNOPSIS**

```
long table( ... ).col( ... ).bytes() const;
```

**DESCRIPTION**

カラムの型が FITS::ASCII_T 以外の場合，カラムの型のバイト数を返します．例えば，カラムの型が FITS::DOUBLE_T の場合，sizeof(fits::double_t) が返ります．カラムの型が FITS::BIT_T の場合は，1 が返ります．

カラムの型が FITS::ASCII_T の場合は，TFORM$n$ と TDIM$n$ 指定によるカラム中の最小要素の文字列長を返します．具体的には次の表のようになります．

| TFORM$n$ と TDIM$n$ の指定 | .bytes() | .dcol_length() | .drow_length() | .elem_length() |
|---|---|---|---|---|
| TFORM$n$ = '120A' | 120 | 1 | 1 | 1 |
| TFORM$n$ = '120A10' | 10 | 12 | 1 | 12 |
| TFORM$n$ = '120A10'<br>TDIM$n$ = '(6,2)' | 10 | 6 | 2 | 12 |
| TFORM$n$ = '120A'<br>TDIM$n$ = '(10,6,2)' | 10 | 6 | 2 | 12 |

**RETURN VALUE**

カラムの型のバイト数を返します．

**EXAMPLES**

§12.8.8の EXAMPLES を参照してください．

---

### 12.8.12 table().col().elem_byte_length()

**NAME**

table().col().elem_byte_length() — カラムの 1 行のバイト数

**SYNOPSIS**

```
long table( ... ).col( ... ).elem_byte_length() const;
```

**DESCRIPTION**

カラムの 1 行のバイト数を返します．例えば，TTYPE$n$が 16D の場合，sizeof(fits::double_t)*16 が返ります．

**RETURN VALUE**

カラムの 1 行のバイト数を返します．

**EXAMPLES**

§12.8.8の EXAMPLES を参照してください．

---

### 12.8.13 table().col().elem_length()

**NAME**

table().col().elem_length() — カラムの 1 行の要素の個数

**SYNOPSIS**

```
long table( ... ).col( ... ).elem_length() const;
```

**DESCRIPTION**

カラムの 1 行の要素の個数を返します.

カラムの型が FITS::ASCII_T 以外の場合，例えば TTYPE$n$ が 16D の場合，16 が返ります (TDIM$n$ とは無関係).

カラムの型が FITS::ASCII_T の場合は，table().col().bytes() メンバ関数 (§12.8.11) の項目にある表を参照してください.

**RETURN VALUE**

カラムの 1 行の要素数を返します.

**EXAMPLES**

§12.8.8の EXAMPLES を参照してください.

---

### 12.8.14 table().col().dcol_length()

**NAME**

table().col().dcol_length() — TDIM$n$ 指定における，1 行の要素数

**SYNOPSIS**

```
long table( ... ).col( ... ).dcol_length() const;
```

**DESCRIPTION**

カラムの TDIM$n$ 指定における，1 行の要素数 (1 次元目の個数) を返します.

カラムの型が FITS::ASCII_T 以外の場合，例えば TDIM$n$ が (8x2) の場合，8 が返ります.

カラムの型が FITS::ASCII_T の場合は，table().col().bytes() メンバ関数 (§12.8.11) の項目にある表を参照してください.

**RETURN VALUE**

TDIM$n$ 指定における，1 行の要素数を返します.

**EXAMPLES**

```
long dcol_count = fits.table("EVENT").col(0L).dcol_length();
```

---

### 12.8.15 table().col().drow_length()

**NAME**

table().col().drow_length() — TDIM$n$ 指定における行数

**SYNOPSIS**

```
long table( ... ).col( ... ).drow_length() const;
```

**DESCRIPTION**

カラムの TDIM$n$ 指定における，行数 (2 次元目以降の個数) を返します.

カラムの型が FITS::ASCII_T 以外の場合，例えば TDIM$n$ が (8x2) の場合，2 が返ります.

カラムの型が FITS::ASCII_T の場合は，table().col().bytes() メンバ関数 (§12.8.11) の項目にある表を参照してください.

**RETURN VALUE**

TDIM$n$ 指定における，列数を返します．

**EXAMPLES**

```
long drow_count = fits.table("EVENT").col(0L).drow_length();
```

---

### 12.8.16 table().col().heap_bytes()

**NAME**

table().col().heap_bytes() — ヒープ (可変長配列) の型のバイト数

**SYNOPSIS**

```
long table( ... ).col( ... ).heap_bytes() const;
```

**DESCRIPTION**

ヒープ (可変長配列) の型のバイト数を返します．例えば，ヒープの型が FITS::DOUBLE_T の場合，sizeof(fits::double_t) が返ります．ヒープの型が FITS::BIT_T の場合は，1 が返ります．

---

### 12.8.17 table().col().max_array_length()

**NAME**

table().col().max_array_length() — 可変長配列の最大の長さ

**SYNOPSIS**

```
long table( ... ).col( ... ).max_array_length() const;
```

**DESCRIPTION**

指定されたカラムにおける，可変長配列の最大の長さを返します．

---

### 12.8.18 table().col().array_length()

**NAME**

table().col().array_length() — 可変長配列の長さ

**SYNOPSIS**

```
long table( ... ).col( ... ).array_length( long row_idx, long elem_idx = 0 ) const;
```

**DESCRIPTION**

指定されたカラムと行における，可変長配列の長さを返します．

エラーの場合は負の値を返します．

---

### 12.8.19 table().col().definition()

**NAME**

table().col().definition() — カラムの定義

**SYNOPSIS**

```
const fits::table_def &table( ... ).col( ... ).definition() const;
```

**DESCRIPTION**

`table().col().definition()` メンバ関数は，カラム定義の構造体オブジェクトへの参照を返します (改変不可)．カラムの定義を別のカラムにコピーする等の場合に利用します．

**RETURN VALUE**

内部に保持されている table_def オブジェクトの参照を返します．

**EXAMPLES**

§12.8.46の EXAMPLES を参照してください．

---

### 12.8.20 table().col().dvalue()

**NAME**

table().col().dvalue() — セルの値を実数値で返す

**SYNOPSIS**

```
double table( ... ).col( ... ).dvalue( long row_index ) const;
double table( ... ).col( ... )
                   .dvalue( long row_index,
                            const char *elem_name, long repetition_idx = 0 ) const;
double table( ... ).col( ... )
                   .dvalue( long row_index,
                            long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの値に，ヘッダの TZERO$n$ と TSCAL$n$ を反映した実数値を返します．NULL 値の場合は，NAN を返します．バイナリテーブルの整数型カラムやアスキーテーブルの TNULL$n$ の値に，セルの値が一致した場合も NULL 値とします．

ヘッダの TZERO$n$ と TSCAL$n$ の値が有効なのは，バイナリテーブルの TFORM$n$ の指定に'B'，'I'，'J'，'K'，'E'，'D' を含む場合と，アスキーテーブルの TFORM$n$ の指定に'I'，'L'，'F'，'E'，'G'，'D' を含む場合です[39)]．

論理型のカラムの場合は，値が'T' なら 1 を，値が'F' なら 0 を，それ以外の場合は NAN を返します．

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，セルの文字列を実数値に変換した値をそのまま返します．

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，セルの文字列を実数値に変換し，その値を TZERO$n$ と TSCAL$n$ で変換した値を返します．

[39)] 'L'，'G' は FITS 規約ではありませんが，SFITSIO ではサポートします．

セルの文字列は，空白文字を除去してから libc の atof() 関数で変換するので，変換できる文字列は 10 進数または 16 進数の整数表現と実数表現です.

NULL 値の場合や，引数が不正な場合は，NAN を返します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します. elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します.

**EXAMPLES**

次のコードは，テーブル「EVENT」のカラム「TIME」のセルの値をすべて表示します.

```
fits_table_col &col_ref = fits.table("EVENT").col("TIME");
long i;
for ( i=0 ; i < col_ref.length() ; i++ ) {
    printf("%f\n",col_ref.dvalue(i));
}
```

---

### 12.8.21 table().col().lvalue(), table().col().llvalue()

**NAME**

table().col().lvalue(), table().col().llvalue() — セルの値を整数値で返す

**SYNOPSIS**

```
long table( ... ).col( ... ).lvalue( long row_index ) const;
long table( ... ).col( ... )
                 .lvalue( long row_index,
                          const char *elem_name, long repetiti_idx = 0 ) const;
long table( ... ).col( ... )
                 .lvalue( long row_index,
                          long elem_index, long repetition_idx = 0 ) const;
long long table( ... ).col( ... ).llvalue( long row_index ) const;
long long table( ... ).col( ... )
                      .llvalue( long row_index,
                                const char *elem_name, long repetiti_idx = 0 ) const;
```

```
long long table( ... ).col( ... )
                      .llvalue( long row_index,
                                long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの値に，ヘッダの TZERO$n$ と TSCAL$n$ を反映した実数値に最も近い整数値を返します．NULL 値の場合は，INDEF_LONG または INDEF_LLONG を返します．バイナリテーブルの整数型カラムやアスキーテーブルの TNULL$n$ の値に，セルの値が一致した場合も NULL 値とします．

ヘッダの TZERO$n$ と TSCAL$n$ の値が有効なのは，バイナリテーブルの TFORM$n$ の指定に'B'，'I'，'J'，'K'，'E'，'D' を含む場合と，アスキーテーブルの TFORM$n$ の指定に'I'，'L'，'F'，'E'，'G'，'D' を含む場合です[40)]．

論理型のカラムの場合は，値が'T' なら 1 を，値が'F' なら 0 を，それ以外の場合は INDEF_LONG または INDEF_LLONG を返します．

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，セルの文字列を実数値に変換し，その値に最も近い整数値を返します．

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，セルの文字列を実数値に変換し，その値を TZERO$n$ と TSCAL$n$ で変換した値に最も近い整数値を返します．

セルの文字列は，空白文字を除去してから libc の atof() 関数で変換するので，変換できる文字列は 10 進数または 16 進数の整数表現と実数表現です．

NULL 値の場合や，引数が不正な場合は，INDEF_LONG または INDEF_LLONG を返します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します．

**EXAMPLES**

§12.8.20の EXAMPLES を参照してください．

---

[40)] 'L'，'G' は FITS 規約ではありませんが，SFITSIO ではサポートします．

### 12.8.22 table().col().bvalue()

**NAME**

table().col().bvalue() — セルの値を論理値で返す

**SYNOPSIS**

```
bool table( ... ).col( ... ).bvalue( long row_index ) const;
bool table( ... ).col( ... )
                 .bvalue( long row_index,
                          const char *elem_name, long repetiti_idx = 0 ) const;
bool table( ... ).col( ... )
                 .bvalue( long row_index,
                          long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの値を論理値で返します．返り値は，true か false です．'T'，'F'，'U' の 3 種類の値を必要とする場合は，table().col().logical_value() メンバ関数 (§12.9.19) を使ってください．

論理型のカラムの場合，値が'T' なら true を，そうでないなら false を返します．

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，セルの文字列を実数値に変換し，その値に最も近い整数値が 0 なら false を，0 でないなら true を返します．

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，セルの文字列を実数値に変換し，さらにその値を TZERO$n$ と TSCAL$n$ で変換し，その値に最も近い整数値が 0 なら false を，0 でないなら true を返します．

セルの文字列は，空白文字を除去してから libc の atof() 関数で変換するので，変換できる文字列は 10 進数または 16 進数の整数表現と実数表現です．セルの文字列が実数に変換できなかった場合，'T' か't' で始まる文字列なら true を，そうでない場合は，false を返します．

整数型や実数型のカラムの場合，セルの値に，ヘッダの TZERO$n$ と TSCAL$n$ を反映した実数値に最も近い整数値が 0 なら false を，0 でないなら true を返します．

NULL 値の場合や，引数が不正な場合は，false を返します．バイナリテーブルの整数型カラムやアスキーテーブルの TNULL$n$ の値に，セルの値が一致した場合も NULL 値とします．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します.

**EXAMPLES**

§12.8.20の EXAMPLES を参照してください.

---

### 12.8.23 table().col().svalue()

**NAME**

table().col().svalue() — セルの値を文字列値で返す

**SYNOPSIS**

```
const char *table( ... ).col( ... ).svalue( long row_index );
const char *table( ... ).col( ... )
                        .svalue( long row_index,
                                 const char *elem_name, long repetiti_idx = 0 );
const char *table( ... ).col( ... )
                        .svalue( long row_index,
                                 long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

セルの値を文字列値で返します.

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，セルの文字列を TFORM$n$ でフォーマットした文字列を返します. TFORM$n$ の指定が無い場合は，セルの文字列をそのまま返します.

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，セルの文字列を実数値に変換し，その値を TZERO$n$ と TSCAL$n$ で変換し，その値を TFORM$n$ でフォーマットした文字列を返します.

論理型のカラムの場合は，TDISP$n$ の指定が無ければ，"T"，"F"，"U"のいずれかを返します. TDISP$n$ の指定が有る場合は，'T' の場合は 1，'F' の場合は 0 を TDISP$n$ でフォーマットした文字列を返します.

整数型や実数型のカラムの場合，セルの値をヘッダの TZERO$n$ と TSCAL$n$ の値で変換し，さらに文字列に変換した値を返します．TDISP$n$ の指定がある場合は，TDISP$n$ の指定に従って変換した文字列を返します.

整数型のカラムで，TZERO$n$ が 0，TSCAL$n$ が 1.0 の場合で，かつ TDISP$n$ の指定が無い場合には，libc の printf() 関数のフォーマット"%lld"に従って変換した文字列を返します.

上記以外の場合には，以下の printf() 関数のフォーマットに従って変換した文字列を返します.

| | | |
|---|---|---|
| FITS::FLOAT_T | … | "%G" |
| FITS::DOUBLE_T, FITS::LONGLONG_T | … | "%.15G" |
| FITS::LONG_T | … | "%.10G" |
| 上記以外のカラムタイプ | … | "%.8G" |

セルが NULL 値を示す場合や，引数が不正な場合は，文字列"NULL"を返します (この文字列は，TDISP$n$ の指定によっては，左右に空白文字が付加される事もあります)．なお，バイナリテー

ブルの整数型カラムやアスキーテーブルの TNULL$n$ の値に，セルの値が一致した場合も NULL 値とします．NULL 文字列値 (デフォルトでは"NULL") は，table().assign_null_svalue() メンバ関数 (§12.8.29) で変更できます．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セル値の文字列のアドレスを返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

§12.8.20の EXAMPLES を参照してください．

---

### 12.8.24 table().col().get_svalue()

**NAME**

table().col().get_svalue() — セルの値を文字列値で得る

**SYNOPSIS**

```
ssize_t table( ... ).col( ... ).get_svalue( long row_index,
                                    char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_svalue( long row_index,
                                    const char *elem_name,
                                    char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_svalue( long row_index,
                                    const char *elem_name, long repetition_idx,
                                    char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_svalue( long row_index,
                                    long elem_index,
                                    char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_svalue( long row_index,
                                    long elem_index, long repetition_idx,
```

```
                                        char *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

セルの値を文字列値で dest_buf に返します.

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，セルの文字列を TFORM$n$ でフォーマットした文字列を返します. TFORM$n$ の指定が無い場合は，セルの文字列をそのまま返します.

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，セルの文字列を実数値に変換し，その値を TZERO$n$ と TSCAL$n$ で変換し，その値を TFORM$n$ でフォーマットした文字列を返します.

論理型のカラムの場合は，TDISP$n$ の指定が無ければ，"T"，"F"，"U"のいずれかを返します. TDISP$n$ の指定が有る場合は，'T' の場合は 1，'F' の場合は 0 を TDISP$n$ でフォーマットした文字列を返します.

整数型や実数型のカラムの場合，セルの値をヘッダの TZERO$n$ と TSCAL$n$ の値で変換し，さらに文字列に変換した値を返します．TDISP$n$ の指定がある場合は，TDISP$n$ の指定に従って変換した文字列を返します.

整数型のカラムで，TZERO$n$ が 0，TSCAL$n$ が 1.0 の場合で，かつ TDISP$n$ の指定が無い場合には，libc の printf() 関数のフォーマット"%lld"に従って変換した文字列を返します.

上記以外の場合には，以下の printf() 関数のフォーマットに従って変換した文字列を返します.

| | | |
|---|---|---|
| FITS::FLOAT_T | … | "%G" |
| FITS::DOUBLE_T, FITS::LONGLONG_T | … | "%.15G" |
| FITS::LONG_T | … | "%.10G" |
| 上記以外のカラムタイプ | … | "%.8G" |

セルが NULL 値を示す場合や，引数が不正な場合は，文字列"NULL"を返します (この文字列は，TDISP$n$ の指定によっては，左右に空白文字が付加される事もあります)．なお，バイナリテーブルの整数型カラムやアスキーテーブルの TNULL$n$ の値に，セルの値が一致した場合も NULL 値とします．NULL 文字列値 (デフォルトでは"NULL") は，table().assign_null_svalue() メンバ関数 (§12.8.29) で変更できます.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します. elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |
| [O] | dest_buf | セルの文字列値取得領域アドレス |
| [I] | buf_size | dest_buf のバイトサイズ |

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : バッファ長が十分な場合にコピーできる文字数 ('\0' は含まない).
負の値 (エラー) : 引数が不正でコピーされなかった場合.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
char buf[128];
fits.table("EVENT").col("TIME").get_svalue( 0, buf, sizeof(buf) );
```

---

### 12.8.25 table().col().assign()

**NAME**

table().col().assign() — 実数値でセルに値を代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... )
                            .assign( double value, long row_index );
fits_table_col &table( ... ).col( ... )
                            .assign( double value, long row_index,
                                     const char *elem_name,
                                     long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... )
                            .assign( double value, long row_index,
                                     long elem_index, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... )
                            .assign( float value, long row_index );
fits_table_col &table( ... ).col( ... )
                            .assign( float value, long row_index,
                                     const char *elem_name,
                                     long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... )
                            .assign( float value, long row_index,
                                     long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

実数 value から，ヘッダの TZERO$n$ と TSCAL$n$ を反映した実数値を求め，セルに代入します. value に NAN を与えた場合は，NULL 値が与えられたものとします．この時，バイナリテーブルの整数型カラムやアスキーテーブルの場合に TNULL$n$ の値があればその値をセルに代入します.

ヘッダの TZERO$n$ と TSCAL$n$ の値が有効なのは，バイナリテーブルの TFORM$n$ の指定に'B', 'I', 'J', 'K', 'E', 'D' を含む場合と，アスキーテーブルの TFORM$n$ の指定に'I', 'L', 'F', 'E', 'G', 'D' を含む場合です[41].

[41] 'L', 'G' は FITS 規約ではありませんが，SFITSIO ではサポートします.

整数型のカラムの場合は，TZERO$n$ と TSCAL$n$ の変換を行なった後，最も近い整数値をセルに代入します．

論理型のカラムの場合は，value に最も近い整数値が 0 なら'F' を，0 以外の値なら'T' をセルに代入します．NULL 値 (NAN) が与えられた時は'\0' を代入します．

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，printf() のフォーマット"%.15G"の形式 (value が double 型の場合) または"%G"の形式 (value が float 型の場合) で変換した文字列をセルに代入します (TFORM$n$ の指定があった場合は，その文字列をさらにフォーマット変換したものを代入します)．

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，value を TZERO$n$ と TSCAL$n$ で変換を行なった後，TFORM$n$ のフォーマットの形式で変換した文字列をセルに代入します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

引数が不正な場合は何もしません．

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
double value = 0;
fits.table("EVENT").col("TIME").assign(value, 0);
```

---

### 12.8.26 table().col().assign()

**NAME**

table().col().assign() — 整数値でセルに値を代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... )
```

```
                              .assign( int value, long row_index );
    fits_table_col &table( ... ).col( ... )
                              .assign( int value, long row_index,
                                       const char *elem_name,
                                       long repetition_idx = 0 );
    fits_table_col &table( ... ).col( ... )
                              .assign( int value, long row_index,
                                       long elem_index, long repetition_idx = 0 );
    fits_table_col &table( ... ).col( ... )
                              .assign( long value, long row_index );
    fits_table_col &table( ... ).col( ... )
                              .assign( long value, long row_index,
                                       const char *elem_name,
                                       long repetition_idx = 0 );
    fits_table_col &table( ... ).col( ... )
                              .assign( long value, long row_index,
                                       long elem_index, long repetition_idx = 0 );
    fits_table_col &table( ... ).col( ... )
                              .assign( long long value, long row_index );
    fits_table_col &table( ... ).col( ... )
                              .assign( long long value, long row_index,
                                       const char *elem_name,
                                       long repetition_idx = 0 );
    fits_table_col &table( ... ).col( ... )
                              .assign( long long value, long row_index,
                                       long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

整数 value から，ヘッダの TZERO$n$ と TSCAL$n$ を反映した実数値を求め，セルに代入します．これらの関数では NULL 値を与える事はできません．NULL 値を与える場合は，double 型か float 型で §12.8.25 のメンバ関数に NAN を指定します．

ヘッダの TZERO$n$ と TSCAL$n$ の値が有効なのは，バイナリテーブルの TFORM$n$ の指定に'B'，'I'，'J'，'K'，'E'，'D' を含む場合と，アスキーテーブルの TFORM$n$ の指定に'I'，'L'，'F'，'E'，'G'，'D' を含む場合です[42]．

整数型のカラムの場合は，TZERO$n$ と TSCAL$n$ の変換を行なった後，最も近い整数値をセルに代入します．

論理型のカラムの場合は，value が 0 なら'F' を，0 以外の値なら'T' をセルに代入します．

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，printf() のフォーマット"%lld"の形式で変換した文字列をセルに代入します (TFORM$n$ の指定があった場合は，その文字列をさらにフォーマット変換したものを代入します)．

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，value を TZERO$n$ と

[42] 'L'，'G' は FITS 規約ではありませんが，SFITSIO ではサポートします．

TSCAL$n$で変換を行なった後，TFORM$n$のフォーマットの形式で変換した文字列をセルに代入します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

引数が不正な場合は何もしません．

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

§12.8.25の EXAMPLES を参照してください．

---

### 12.8.27 table().col().assign()

**NAME**

table().col().assign() — 文字列値でセルに値を代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... )
                            .assign( const char *value, long row_index );
fits_table_col &table( ... ).col( ... )
                            .assign( const char *value, long row_index,
                                     const char *elem_name,
                                     long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... )
                            .assign( const char *value, long row_index,
                                     long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

文字列値でセルに値を代入します．value に NULL か文字列"NULL"(文字列の左右に空白があっても良い) が与えられた場合は，NULL 値が与えられたものとします．この時，バイナリテー

ブルの整数型カラムやアスキーテーブルの場合に TNULL$n$ の値があればその値をセルに代入します．NULL 文字列値 (デフォルトでは"NULL") は，table().assign_null_svalue() メンバ関数 (§12.8.29) で変更できます.

バイナリテーブルの文字列型のカラムの場合と，アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示さない場合には，文字列値 value を TFORM$n$ でフォーマットした文字列を代入します．TFORM$n$ の指定が無い場合は，文字列値 value をそのままセルに代入します.

アスキーテーブルの当該カラムの TFORM$n$ が数値表現を示す場合には，文字列 value を実数値に変換し，さらにその値を TZERO$n$ と TSCAL$n$ の値で変換した値を，TFORM$n$ でフォーマット変換した文字列をセルに代入します．文字列 value は，空白文字を除去してから libc の atof() 関数で変換するので，変換できる文字列は 10 進数または 16 進数の整数表現と実数表現です.

論理型のカラムの場合は，文字列 value を実数値に変換できる場合は，それぞれ，0 でない値，0，NaN の場合に，'T'，'F'，'\0' をセルに代入します．実数値に変換できない場合は，value が'T' か't' で始まる文字列なら'T' を，'F' か'f' で始まる文字列なら'F' を，そうでない場合は，'\0' を代入します.

整数型や実数型のカラムの場合，文字列 value を実数値に変換し，ヘッダの TZERO$n$ と TSCAL$n$ の値で変換した実数値をセルに代入します (カラムが整数型の場合は，最も近い整数が代入されます).

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

引数が不正な場合は何もしません.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.8.25の EXAMPLES を参照してください.

### 12.8.28 table().col().convert_type()

**NAME**

table().col().convert_type() — データ型の変換・変更

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).convert_type( int new_type );
fits_table_col &table( ... ).col( ... ).convert_type( int new_type,
                                                      double new_zero );
fits_table_col &table( ... ).col( ... ).convert_type( int new_type,
                                                      double new_zero,
                                                      double new_scale );
fits_table_col &table( ... ).col( ... ).convert_type( int new_type,
                                                      double new_zero,
                                                      double new_scale,
                                                      double new_null );
```

**DESCRIPTION**

整数型あるいは実数型のカラム (TFORM$n$ の指定が'B', 'I', 'J', 'K', 'E', 'D' を含む場合) の型を new_type に変更します. 必要に応じて, 内部バッファのサイズも変更します. new_type に指定できる値は, FITS::DOUBLE_T, FITS::FLOAT_T, FITS::LONGLONG_T, FITS::LONG_T, FITS::SHORT_T, FITS::BYTE_T のいずれかです. new_zero, new_scale, new_null を指定した場合は, ヘッダの TZERO$n$, TSCAL$n$, TNULL$n$ も変更し, それらの値を反映したデータに変換します. 引数 new_null が有効なのは, new_type が整数型の場合のみです.

文字列型や論理型のカラムはこのメンバ関数では変換できません.

**PARAMETER**

[I] new_type 変換後のデータ型
[I] new_zero 変換後の TZERO 値
[I] new_scale 変換後の TSCAL 値
[I] new_null 変換後の TNULL 値

([I]: 入力, [O]: 出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば, 変換後のテーブルが拡大する際にテーブル領域の再確保で失敗した場合), SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").col("PIX_DATA").convert_type(FITS::DOUBLE_T);
```

---

### 12.8.29 table().assign_null_svalue()

**NAME**

table().assign_null_svalue() — ヌル文字列値の操作

**SYNOPSIS**

```
fits_table &table( ... ).assign_null_svalue( const char *snull );
```

**DESCRIPTION**

table().assign_null_svalue() メンバ関数は，table().col().svalue()(§12.8.23) table().col().assign()(
で使用する NULL 文字列値を設定します．

デフォルトでは，svalue() と assign() では "NULL" がヌル文字列として使用されますが，assign_null_svalue() メンバ関数を使う事でヌル文字列を変更する事ができます．

**PARAMETER**

[I] snull セットする NULL 文字列
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

---

### 12.8.30 table().col().tzero(), table().col().assign_tzero()

**NAME**

table().col().tzero(), table().col().assign_tzero() — ゼロ点の操作

**SYNOPSIS**

```
double table( ... ).col( ... ).tzero() const;
bool table( ... ).col( ... ).tzero_is_set() const;
fits_table_col &table( ... ).col( ... ).assign_tzero( double zero, int prec = 15 );
fits_table_col &table( ... ).col( ... ).erase_tzero();
```

**DESCRIPTION**

table().col().tzero() メンバ関数は，TZERO$n$ の値を返します．

table().col().assign_tzero() メンバ関数は，TZERO$n$ の値を設定します．prec には桁数を指定できます．省略した場合，15 桁の精度でヘッダレコードに書き込みます．

table().col().erase_tzero() メンバ関数は，TZERO$n$ の設定を消去します．

**PARAMETER**

[I] zero セットする TZERO 値
[I] prec 精度 (桁数)
([I]：入力，[O]：出力)

**RETURN VALUE**

tzero() メンバ関数は TZERO$n$ の値を返します．
tzero_is_set() メンバ関数は TZERO$n$ の定義の有無を返します．
assign_tzero(),erase_tzero() メンバ関数は，当該 fits_table_col オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
if ( fits.table("EVENT").col("PIX_DATA").tzero_is_set() == false ) {
    fits.table("EVENT").col("PIX_DATA").assign_tzero(0.0);
}
```

---

### 12.8.31 table().col().tscal(), table().col().assign_tscal()

**NAME**

table().col().tscal(), table().col().assign_tscal() — スケーリングファクターの操作

**SYNOPSIS**

```
double table( ... ).col( ... ).tscal() const;
bool table( ... ).col( ... ).tscal_is_set() const;
fits_table_col &table( ... ).col( ... ).assign_tscal( double scal, int prec = 15 );
fits_table_col &table( ... ).col( ... ).erase_tscal();
```

**DESCRIPTION**

table().col().tscal() メンバ関数は，`TSCAL`$n$ の値を返します.

table().col().assign_tscal() メンバ関数は，`TSCAL`$n$ の値を設定します．`prec` には桁数を指定できます．省略した場合，15 桁の精度でヘッダレコードに書き込みます.

table().col().erase_tscal() メンバ関数は，`TSCAL`$n$ の設定を消去します.

**PARAMETER**

[I] `scal` セットする `TSCAL` 値
[I] `prec` 精度 (桁数)
([I]：入力，[O]：出力)

**RETURN VALUE**

tscal() メンバ関数は `TSCAL`$n$ の値を返します.
tscal_is_set() メンバ関数は `TSCAL`$n$ の定義の有無を返します.
assign_tscal(),erase_tscal() メンバ関数は, 当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
if ( fits.table("EVENT").col("PIX_DATA").tscal_is_set() == false ) {
    fits.table("EVENT").col("PIX_DATA").assign_tscal(1.0);
}
```

---

### 12.8.32 table().col().tnull(), table().col().assign_tnull()

**NAME**

table().col().tnull(), table().col().assign_tnull() — ヌル値の操作

**SYNOPSIS**

```
long long table( ... ).col( ... ).tnull( const char **tnull_ptr = NULL ) const;
bool table( ... ).col( ... ).tnull_is_set() const;
fits_table_col &table( ... ).col( ... ).assign_tnull( long long null );
fits_table_col &table( ... ).col( ... ).erase_tnull();
```

**DESCRIPTION**

table().col().tnull() メンバ関数は，TNULL$n$ の値を返します．Ascii Table の場合で，文字列の TNULL 値が必要な場合は，tnull_ptr で内部バッファの文字列のアドレスを取得する事ができます．

table().col().assign_tnull() メンバ関数は，TNULL$n$ の値を設定します．

table().col().erase_tnull() メンバ関数は，TNULL$n$ の設定を消去します．

**PARAMETER**

[I] null セットする TNULL 値
[O] tnull_ptr 文字列型の TNULL 値のアドレス (Ascii Table 時のみ利用)
([I]：入力，[O]：出力)

**RETURN VALUE**

tnull() メンバ関数は TNULL$n$ の値を返します．
tnull_is_set() メンバ関数は TNULL$n$ の定義の有無を返します．
assign_tnull()，erase_tnull() メンバ関数は，当該 fits_table_col オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
if ( fits.table("EVENT").col("PIX_DATA").tnull_is_set() == false ) {
    fits.table("EVENT").col("PIX_DATA").assign_tnull(-1);
}
```

---

### 12.8.33 table().col().tunit(), table().col().assign_tunit()

**NAME**

table().col().tunit(), table().col().assign_tunit() — 物理単位の操作

**SYNOPSIS**

```
const char *table( ... ).col( ... ).tunit() const;
bool table( ... ).col( ... ).tunit_is_set() const;
fits_table_col &table( ... ).col( ... ).assign_tunit( const char *unit );
fits_table_col &table( ... ).col( ... ).erase_tunit();
```

**DESCRIPTION**

table().col().tunit() メンバ関数は，TUNIT$n$ の値を返します.

table().col().assign_tunit() メンバ関数は，TUNIT$n$ の値を設定します.

table().col().erase_tunit() メンバ関数は，TUNIT$n$ の設定を消去します.

**PARAMETER**

[I] `unit` セットする TUNIT 値

([I]：入力，[O]：出力)

**RETURN VALUE**

tunit() メンバ関数は TUNIT$n$ の値を返します.

tunit_is_set() メンバ関数は TUNIT$n$ の定義の有無を返します.

assign_tunit(),erase_tunit() メンバ関数は, 当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
if ( fits.table("EVENT").col("RA").tunit_is_set() == false ) {
    fits.table("EVENT").col("RA").assign_tunit("deg");
}
```

---

### 12.8.34 table().init()

**NAME**

table().init() — テーブルの初期化

**SYNOPSIS**

```
fits_table &table( ... ).init();
fits_table &table( ... ).init( const fits::table_def defs[] );
```

**DESCRIPTION**

ヘッダとテーブルの内容をすべて消去し，初期化します.

`defs` が指定されれば場合には，それに従ってカラムを作成します.

**PARAMETER**

[I] `defs` `fits::table_def` 構造体

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").init();
```

---

### 12.8.35 table().col().init()

**NAME**

table().col().init() — カラムの初期化

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).init();
fits_table_col &table( ... ).col( ... ).init( const fits_table_col &src );
```

**DESCRIPTION**

カラムの内容をすべて消去し，初期化します.

src が指定された場合には，src の内容をコピーしたカラムで上書きします．src の行数に関係なく，テーブルの行数はメンバ関数呼び出し前と変わりません (src の行数がテーブルの行数に比べて長い場合，後部のセルの内容はコピーされません).

**PARAMETER**

[I] src 源泉となるカラムオブジェクトの参照
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードの場合，“EVENT” テーブルのカラム “LON” の名前は “RA” に変わります.

```
fits.table("EVENT").col("LON").init( fits_r.table("RAW").col("RA") );
```

---

### 12.8.36 table().ascii_to_binary()

**NAME**

table().ascii_to_binary() — アスキーテーブルをバイナリテーブルに変換

**SYNOPSIS**

```
fits_table &table( ... ).ascii_to_binary();
```

**DESCRIPTION**

当該テーブルがアスキーテーブルの属性となっていた場合，属性をバイナリテーブルに変換します.

属性をバイナリテーブルにすると，FITS ファイル上のアスキーテーブルの TFORM$n$ の値 (fits::table_def 構造体の tdisp メンバの値) は，バイナリテーブルとして保存される時には TFORM$n$ のコメント部分に保存されます.

TNULL$n$ の値も，TNULL$n$ のコメント部分に保存されます (値は未定義となります).

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits.table("X_CATALOG").ascii_to_binary();
```

---

### 12.8.37 table().assign_col_name()

**NAME**

table().assign_col_name() — カラム名の設定

**SYNOPSIS**

```
fits_table &table( ... ).assign_col_name( long col_index, const char *newname );
fits_table &table( ... ).assign_col_name( const char *col_name, const char *newname );
```

**DESCRIPTION**

col_index あるいは col_name で示されたカラムの名前を newname に設定します.

**PARAMETER**

[I] col_index カラムインデックス
[I] col_name カラム名
[I] newname 新しく設定するカラム名
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").assign_col_name(0L, "TIME");
```

---

### 12.8.38 table().define_a_col()

**NAME**

table().define_a_col() — カラムの定義の変更

**SYNOPSIS**

```
fits_table &table( ... ).define_a_col( long col_index,
                                       const fits::table_def &def );
fits_table &table( ... ).define_a_col( const char *col_name,
                                       const fits::table_def &def );
```

**DESCRIPTION**

col_index あるいは col_name で示されたカラムの定義を変更します.

設定を変更したくない項目については，def のメンバに NULL を代入します.

**PARAMETER**

[I] col_index カラムインデックス
[I] col_name カラム名
[I] defs fits::table_def 構造体
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits::table_def def =
    { "TIME","satellite time", NULL,NULL, "s","", "F16.3", "1D", "" };
fits.table("EVENT").define_a_col(0L, def);
```

---

### 12.8.39 table().col_header_index()

**NAME**

table().col_header_index() — 指定されたカラム定義のヘッダレコードの番号

**SYNOPSIS**

```
long table( ... ).col_header_index( const char *col_name,
                                    const char *kwd ) const;
long table( ... ).col_header_index( long col_index,
                                    const char *kwd ) const;
```

**DESCRIPTION**

kwd で指定されたカラムキーワードのプレフィックス (例: TTYPE) を持つヘッダレコードのうち，col_index あるいは col_name で示されたカラムに対応するヘッダレコードの番号を返します.

**PARAMETER**

[I] col_index カラムインデックス
[I] col_name カラム名
[I] kwd カラムキーワードのプレフィックス
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : レコード番号.
負の値 (エラー) : 見つからなかった場合.

**EXAMPLES**

```
long idx = fits.table("EVENT").col_header_index("DATE", "TTYPE");
```

---

### 12.8.40 table().col_header()

**NAME**

table().col_header() — カラムの定義のヘッダレコードへの参照

**SYNOPSIS**

```
fits_header_record &table( ... ).col_header( const char *col_name,
                                             const char *kwd );
fits_header_record &table( ... ).col_header( long col_index,
                                             const char *kwd );
const fits_header_record &table( ... ).col_header( const char *col_name,
                                                   const char *kwd ) const;
const fits_header_record &table( ... ).col_header( long col_index,
                                                   const char *kwd ) const;
```

**DESCRIPTION**

kwd で指定されたカラムキーワードのプレフィックス (例: TTYPE) を持つヘッダレコードのうち，col_index あるいは col_name で示されたカラムに対応するヘッダレコードの参照を返します.

.col_header() に続くメンバ関数として，.svalue()，.dvalue()，.lvalue() 等が使えます. これらについては，§12.4.4以降を参照してください.

**PARAMETER**

[I] col_index カラムインデックス
[I] col_name カラム名
[I] kwd カラムキーワードのプレフィックス
([I]：入力，[O]：出力)

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．読み取り専用のオブジェクトにおいて存在しないカラムキーワードを指定した場合に，SFITSIO に由来する例外が発生します.

**EXAMPLES**

```
printf("TTYPE of 'TIME' = %s\n",
       fits.table("EVENT").col_header("TIME","TTYPE").svalue());
```

---

### 12.8.41 table().update_col_header()

**NAME**

table().update_col_header() — カラムの定義のヘッダレコードの更新

**SYNOPSIS**

```
fits_table &table( ... ).update_col_header( const char *col_name,
                           const char *kwd, const char *val, const char *com );
fits_table &table( ... ).update_col_header( long col_index,
                           const char *kwd, const char *val, const char *com );
```

**DESCRIPTION**

kwd で指定されたカラムキーワードのプレフィックス (例: TTYPE) を持つヘッダレコードのうち，col_index あるいは col_name で示されたカラムに対応するヘッダレコードを更新し，同時にオブジェクト内部の管理情報も更新します.

**PARAMETER**

| | | |
|---|---|---|
| [I] | col_index | カラムインデックス |
| [I] | col_name | カラム名 |
| [I] | kwd | カラムキーワードのプレフィックス |
| [I] | val | ヘッダレコードの値 |
| [I] | com | ヘッダレコードのコメント |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").update_col_header("TIME","TUNIT","s","unit");
```

---

### 12.8.42 table().erase_col_header()

**NAME**

table().erase_col_header() — カラムの定義のヘッダレコードの削除

**SYNOPSIS**

```
fits_table &table( ... ).erase_col_header( const char *col_name,
                                           const char *kwd );
fits_table &table( ... ).erase_col_header( long col_index,
                                           const char *kwd );
```

**DESCRIPTION**

kwd で指定されたカラムキーワードのプレフィックス (例: TTYPE) を持つヘッダレコードのうち，col_index あるいは col_name で示されたカラムに対応するヘッダレコードを削除します. 同時にオブジェクト内部の管理情報も更新します.

**PARAMETER**

[I] `col_index` カラムインデックス
[I] `col_name` カラム名
[I] `kwd` カラムキーワードのプレフィックス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").erase_col_header("TIME","TUNIT");
```

---

### 12.8.43 table().rename_col_header()

**NAME**

table().rename_col_header() — ユーザ定義のカラム用ヘッダキーワードの名前を変更する

**SYNOPSIS**

```
fits_table &table( ... ).rename_col_header( const char *old_kwd,
                                            const char *new_kwd );
```

**DESCRIPTION**

ユーザ定義のカラム用ヘッダレコードのキーワードプレフィックスを，`old_kwd` から `new_kwd` へ名前を変更します．FITS 規約で定められたキーワード (`TTYPE` 等) については変更できません.

**PARAMETER**

[I] `old_kwd` 変更前のカラムキーワードのプレフィックス
[I] `old_kwd` 変更後のカラムキーワードのプレフィックス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードは，ユーザ定義のカラムキーワード `TLMAX`$n$ を `TMMAX`$n$ に変更します.

```
fits.table("EVENT").rename_col_header("TLMAX","TMMAX");
```

---

### 12.8.44 table().sort_col_header()

**NAME**

table().sort_col_header() — カラム用ヘッダキーワードをカラム順にソートする

**SYNOPSIS**

```
fits_table &table( ... ).sort_col_header();
```

**DESCRIPTION**

すべてのカラム用ヘッダレコードを，カラム順にソートします.

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").sort_col_header();
```

---

### 12.8.45 table().swap()

**NAME**

table().swap() — テーブルの交換

**SYNOPSIS**

```
fits_table &table( ... ).swap( fits_table &obj );
```

**DESCRIPTION**

自身と `obj` との中身を交換します.

**PARAMETER**

[I/O] `obj` 入れ替え対象のオブジェクト
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードは，オブジェクト fits 内のテーブル「EVENT」とテーブル「EVENT_SAVE」の中身を交換します.

```
fits.table("EVENT").swap(fits.table("EVENT_SAVE"));
```

---

### 12.8.46 table().append_cols(), table().append_a_col()

**NAME**

table().append_cols(), table().append_a_col() — カラムの追加

**SYNOPSIS**

```
fits_table &table( ... ).append_cols( const fits::table_def defs[] );
fits_table &table( ... ).append_cols( fits_table &src );
fits_table &table( ... ).append_a_col( const fits::table_def &def );
fits_table &table( ... ).append_a_col( fits_table_col &src );
```

**DESCRIPTION**

テーブルにカラムを追加します．append_cols() は複数のカラムを，append_a_col() は 1 つのカラムを追加します．

src を指定した場合は，src の持つカラムの定義だけでなくデータ領域もコピーしますが，行数が十分でない場合はすべての行がコピーされません．

**PARAMETER**

[I] defs fits::table_def 構造体
[I] src コピー元となるカラムを持つオブジェクト
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，空きメモリ領域に対して大きすぎるカラムを追加しようとした場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
const table_def &def = fits.table("EVENT").col(0L).definition();
fits.table("EVENT_SAVE").append_a_col(def);
```

---

### 12.8.47 table().insert_cols(), table().insert_a_col()

**NAME**

table().insert_cols(), table().insert_a_col() — カラムの挿入

**SYNOPSIS**

```
fits_table &table( ... ).insert_cols( long index, const fits::table_def defs[]);
fits_table &table( ... ).insert_cols( const char *col_name,
                                      const fits::table_def defs[] );
fits_table &table( ... ).insert_cols( long index, fits_table &src );
fits_table &table( ... ).insert_cols( const char *col_name, fits_table &src );
fits_table &table( ... ).insert_a_col( long col_index,
                                       const fits::table_def &def );
```

```
fits_table &table( ... ).insert_a_col( const char *col_name,
                                       const fits::table_def &def );
```

**DESCRIPTION**

index あるいは col_name で指定されたカラムの前に, 新しいカラムを挿入します. insert_cols() は複数のカラムを, insert_a_col() は 1 つのカラムを追加します.

src を指定した場合は, src の持つカラムの定義だけでなくデータ領域もコピーしますが, 行数が十分でない場合はすべての行がコピーされません.

**PARAMETER**

| | | |
|---|---|---|
| [I] | index | 挿入位置を示すカラムインデックス |
| [I] | col_name | 挿入位置を示すカラム名 |
| [I] | defs | fits::table_def 構造体 |
| [I] | src | 挿入されるカラムを持つオブジェクト |

([I]：入力, [O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば, 空きメモリ領域に対して, 大きすぎるカラムを挿入しようとした場合), SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits::table_def def =
    { "TIME_SAVE","saved time", NULL,NULL, "s","", "F16.3", "1D", "" };
fits.table("EVENT").insert_a_col(1, def);
```

---

### 12.8.48 table().swap_cols()

**NAME**

table().swap_cols() — カラムの入れ替え

**SYNOPSIS**

```
fits_table &table( ... ).swap_cols( long index0, long num_cols, long index1 );
fits_table &table( ... ).swap_cols( const char *col_name0, long num_cols,
                                    const char *col_name1 );
```

**DESCRIPTION**

index0 あるいは col_name0 で指定されたカラムから num_cols 個のカラム群を, index1 あるいは col_name1 で指定されたカラムから num_cols 個のカラム群と入れ替えます.

num_cols によって, 2 つのカラム群が重なる場合は, num_cols の値を減らして入れ替えを行ないます.

**PARAMETER**

[I] `index0` 入れ替え元を示すカラムインデックス
[I] `col_name0` 入れ替え元を示すカラム名
[I] `num_cols` 入れ替えるカラム数
[I] `index1` 入れ替え先を示すカラムインデックス
[I] `col_name1` 入れ替え先を示すカラム名
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードは，カラム 0 とカラム 2 の中身を入れ替えます.

```
fits.table("EVENT").swap_cols(0, 1, 2);
```

---

### 12.8.49 table().erase_cols(), table().erase_a_col()

**NAME**

table().erase_a_col() — カラムの消去

**SYNOPSIS**

```
fits_table &table( ... ).erase_cols( long index, long num_cols );
fits_table &table( ... ).erase_cols( const char *col_name, long num_cols );
fits_table &table( ... ).erase_a_col( long col_index );
fits_table &table( ... ).erase_a_col( const char *col_name );
```

**DESCRIPTION**

`index` あるいは `col_name` で指定されたカラムから `num_cols` 個のカラム群を消去します.

**PARAMETER**

[I] `index` 消去開始位置を示すカラムインデックス
[I] `col_name` 消去開始位置を示すカラム名
[I] `num_cols` 消去するカラム数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").erase_cols(0L, 1);
```

---

### 12.8.50 table().copy()

**NAME**

table().copy() — 行の別オブジェクトへのコピー

**SYNOPSIS**

```
void table( ... ).copy( fits_table *dest ) const;
void table( ... ).copy( long idx_begin, long num_rows, fits_table *dest ) const;
```

**DESCRIPTION**

idx_begin で指定された行から num_rows 個の行を，dest で示されたオブジェクトにコピーします.

idx_begin 等の指定がない場合は，すべての行をコピーします.

import_rows() メンバ関数 (§12.8.58) に与える一時バッファを用意する場合などに利用します.

**PARAMETER**

[I] idx_begin コピー元を示す行インデックス
[I] num_rows コピーする行数
[O] dest 指定された行のコピー先となるオブジェクト
([I]：入力，[O]：出力)

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
fits_table tmp_buf;
fits.table("EVENT").copy(0L,40, &tmp_buf);
```

---

### 12.8.51 table().resize_rows()

**NAME**

table().resize_rows() — テーブル行数の変更

**SYNOPSIS**

```
fits_table &table( ... ).resize_rows( long num_rows );
```

**DESCRIPTION**

テーブルの行数を num_rows 行に変更します.

**PARAMETER**

[I] num_rows 変更後の行数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，空きメモリに対して指定された行数が大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").resize_rows(100);
```

---

### 12.8.52 table().append_rows(), table().append_a_row()

**NAME**

table().append_rows(), table().append_a_row() — テーブルの行の追加

**SYNOPSIS**

```
fits_table &table( ... ).append_rows( long num_rows );
fits_table &table( ... ).append_a_row();
```

**DESCRIPTION**

table().append_rows() メンバ関数は，テーブルの最後に，`num_rows` 個の新しい行を追加します.

table().append_a_row() メンバ関数は，テーブルの最後に，1 個の新しい行を追加します.

整数型と実数型のロウの場合，追加された行の値は 0，論理型の場合は`'\0'`，文字列型の場合は`' '`からなる文字列です.

**PARAMETER**

[I] `num_rows` 追加される行数

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，空きメモリに対して指定された行数が大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").append_rows(20);
```

---

### 12.8.53 table().insert_rows(), table().insert_a_row()

**NAME**

table().insert_rows(), table().insert_a_row() — テーブルの行の挿入

**SYNOPSIS**

```
fits_table &table( ... ).insert_rows( long index, long num_rows );
fits_table &table( ... ).insert_a_row( long index );
```

**DESCRIPTION**

table().insert_rows() メンバ関数は，index で指定された行に，num_rows 個の新しい行を挿入します.

table().insert_a_row() メンバ関数は，index で指定された行に，1 個の新しい行を挿入します.

整数型と実数型のカラムの場合，挿入された行の値は 0，論理型の場合は'\0'，文字列型の場合は' 'からなる文字列です.

**PARAMETER**

[I] index 挿入位置を示す行インデックス
[I] num_rows 挿入される行数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，空きメモリに対して指定された行数が大きすぎる場合)，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードはテーブルの 10 行目の後ろに 5 行の新規行を挿入します.

```
fits.table("EVENT").insert_rows(10, 5);
```

---

### 12.8.54 table().erase_rows(), table().erase_a_row()

**NAME**

table().erase_rows(), table().erase_a_row() — テーブルの行の消去

**SYNOPSIS**

```
fits_table &table( ... ).erase_rows( long index, long num_rows );
fits_table &table( ... ).erase_a_row( long index );
```

**DESCRIPTION**

table().erase_rows() メンバ関数は，index で指定された行から num_rows 個の行を削除します.

table().erase_a_row() メンバ関数は，index で指定された行から 1 個の行を削除します.

**PARAMETER**

[I] index 削除開始位置を示す行インデックス
[I] num_rows 削除される行数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

次のコードはテーブルの 10 行目の後ろ 5 行を削除します.

```
fits.table("EVENT").erase_rows(10, 5);
```

---

### 12.8.55 table().clean_rows()

**NAME**

table().clean_rows() — テーブルの行の初期化

**SYNOPSIS**

```
fits_table &table( ... ).clean_rows();
fits_table &table( ... ).clean_rows( long index, long num_rows );
```

**DESCRIPTION**

table().clean_rows() メンバ関数は, すべてのカラムにおいて, index で指定された行から num_rows 個の行を初期値にします. 引数が指定されない場合は, すべての行が対象です.

整数型と実数型のカラムの場合の初期値は 0, 論理型の場合は'\0', 文字列型の場合は' 'からなる文字列です.

**PARAMETER**

[I] index 初期化開始位置を示す行インデックス
[I] num_rows 初期化される行数
([I]：入力, [O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXAMPLES**

次のコードはテーブルの 10 行目から 5 行を初期化します.

```
fits.table("EVENT").clean_rows(10, 5);
```

---

### 12.8.56 table().move_rows()

**NAME**

table().move_rows() — テーブルの行から行へのコピー

**SYNOPSIS**

```
fits_table &table( ... ).move_rows( long src_index, long num_rows, long dest_index );
```

**DESCRIPTION**

すべてのカラムにおいて, src_index で指定された行から num_rows 個の行を, dest_index で指定された行から始まる行へコピーします.

**PARAMETER**

[I] `src_index` コピー元を示す行インデックス
[I] `num_rows` コピーされる行数
[I] `dest_index` コピー先を示す行インデックス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXAMPLES**

次のコードはテーブルの 10 行目を 11 行目にコピーします.

```
fits.table("EVENT").move_rows(10, 1, 11);
```

---

### 12.8.57 table().swap_rows()

**NAME**

table().swap_rows() — テーブルの行と行との入れ替え

**SYNOPSIS**

```
fits_table &table( ... ).swap_rows( long index0, long num_rows, long index1 );
```

**DESCRIPTION**

すべてのカラムにおいて，`index0` で指定された行から `num_rows` 個の行を，`index1` で指定された行から `num_cols` 個の行と入れ替えます.

`num_cols` によって，重なる行がある場合は，`num_cols` の値を減らして入れ替えを行ないます.

**PARAMETER**

[I] `index0` 入れ替え元を示す行インデックス
[I] `num_rows` 入れ替えを行う行数
[I] `index1` 入れ替え先を示す行インデックス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXAMPLES**

次のコードはテーブルの 10 行目を 11 行目に入れ替えます.

```
fits.table("EVENT").swap_rows(10, 1, 11);
```

---

### 12.8.58 table().import_rows()

**NAME**

table().import_rows() — テーブルのインポート

**SYNOPSIS**

```
fits_table &table( ... ).import_rows( long dest_index, bool match_by_name,
                                      const fits_table &from,
                                      long idx_begin = 0,
                                      long num_rows = FITS::ALL );
```

**DESCRIPTION**

テーブルオブジェクト from の idx_begin から num_rows 個の行を，dest_index で指定された行から num_rows 個の行へインポートします．インポートは，すべてのカラムが対象です．

from のそれぞれのカラムを，当該オブジェクトのどのカラムへ割り当てるかは，match_by_name で決めます．match_by_name が true の場合，カラム名が一致するものを探し，一致すればインポートします．match_by_name が false の場合，カラム番号 0 から順にインポートします．

from の持つカラムと当該オブジェクトの持つカラムの型は，一致している必要はありません．一致しない場合は，値を変換してインポートします．

**PARAMETER**

| | | |
|---|---|---|
| [I] | dest_index | インポート先の行インデックス |
| [I] | match_by_name | インポート時のカラム名一致フラグ |
| [I] | from | インポート元のテーブルオブジェクト |
| [I] | idx_begin | インポート元の行インデックス |
| [I] | num_rows | インポートを行う行数 |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
fits.table("EVENT").import_rows( 0, false, fits.table("EVENT_SAVE") );
```

---

### 12.8.59 table().col().move()

**NAME**

table().col().move() — 特定のカラムで，行から行へのコピー

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... )
                            .move( long src_index, long num_rows, long dest_index );
```

**DESCRIPTION**

指定されたカラムにおいて，src_index で指定された行から num_rows 個の行を，dest_index で指定された行から始まる行へコピーします．

**PARAMETER**

[I] `src_index` コピー元を示す行インデックス
[I] `num_rows` コピーを行う行数
[I] `dest_index` コピー先を示す行インデックス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXAMPLES**

次のコードはカラム 0 において，0 行目から 2 行目へ 1 行コピーします.

```
fits.table("EVENT").col(0L).move( 0, 1, 2 );
```

---

### 12.8.60 table().col().swap()

**NAME**

table().col().swap() — 特定のカラムで，行と行との入れ替え

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... )
                            .swap( long index0, long num_rows, long index1 );
```

**DESCRIPTION**

指定されたカラムにおいて，`index0` で指定された行から `num_rows` 個の行を，`index1` で指定された行から `num_cols` 個の行と入れ替えます.

`num_cols` によって，重なる行がある場合は，`num_cols` の値を減らして入れ替えを行ないます.

**PARAMETER**

[I] `index0` 入れ替え元を示す行インデックス
[I] `num_rows` 入れ替えを行う行数
[I] `index1` 入れ替え先を示す行インデックス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXAMPLES**

次のコードはカラム 0 において，0 行目と 2 行目を入れ替えます.

```
fits.table("EVENT").col(0L).swap( 0, 1, 2 );
```

---

### 12.8.61 table().col().clean()

**NAME**

table().col().clean() — 特定のカラムで，値を初期化

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).clean();
fits_table_col &table( ... ).col( ... ).clean( long index, long num_rows );
```

**DESCRIPTION**

指定されたカラムにおいて，index で指定された行から num_rows 個の行を初期値にします．引数が指定されない場合は，すべての行が対象です．

整数型と実数型のカラムの場合の初期値は 0，論理型の場合は'\0'，文字列型の場合は' 'からなる文字列です．

**PARAMETER**

[I] index 初期化の開始位置を示す行インデックス
[I] num_rows 初期化を行う行数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します．

**EXAMPLES**

```
fits.table("EVENT").col(0L).clean();
```

---

### 12.8.62 table().col().import()

**NAME**

table().col().import() — 特定のカラムでのインポート

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... )
                            .import( long dest_index,
                                     const fits_table_col &from,
                                     long idx_begin = 0,
                                     long num_rows = FITS::ALL );
```

**DESCRIPTION**

指定されたカラムにおいて，テーブルカラムオブジェクト from の idx_begin から num_rows 個の行を，dest_index で指定された行から num_rows 個の行へインポートします．

from の持つカラムと当該オブジェクトのカラムの型は，一致している必要はありません．一致しない場合は，値を変換してインポートします．

**PARAMETER**

[I] dest_index インポート先の行インデックス
[I] from インポート元のテーブルオブジェクト
[I] idx_begin インポート元の行インデックス
[I] num_rows インポートを行う行数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB または SFITSIO に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

```
fits.table("EVENT").col(0L).import( 0, fits.table("EVENT_SAVE").col(0L) );
```

---

### 12.8.63 table().col().assign_default()

**NAME**

table().col().assign_default() — 行の長さを大きくした場合の新規セルの値を設定

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_default( double value );
fits_table_col &table( ... ).col( ... ).assign_default( float value );
fits_table_col &table( ... ).col( ... ).assign_default( long long value );
fits_table_col &table( ... ).col( ... ).assign_default( long value );
fits_table_col &table( ... ).col( ... ).assign_default( int value );
fits_table_col &table( ... ).col( ... ).assign_default( const char *value );
fits_table_col &table( ... ).col( ... )
                            .assign_default_value( const void *value_ptr );
```

**DESCRIPTION**

`table().resize_rows()` 等でテーブルの行の長さを大きくした場合の新規セルの値を設定します.

`.assign_default()` は高レベルなメンバ関数で，ヘッダの `TZERO`$n$，`TSCALE`$n$，`TNULL`$n$ の値が反映されます．NULL 値をセットしたい場合は `NAN` をセットします.

`.assign_default_value()` は低レベルなメンバ関数で，ヘッダの `TZERO`$n$ 等の値は考慮されません．オブジェクト内のセルの型に一致する変数または定数のアドレスを与える必要があります.

**PARAMETER**

[I] `value` デフォルト値
[I] `value_ptr` デフォルト値を持つユーザ変数または定数のアドレス
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合，SLLIB に由来する例外 (`sli::err_rec` 例外) が発生します.

**EXAMPLES**

次のコードでは，最初のカラムについて，行の長さを変更した時の新規セルの値を NULL 値に設定し，テーブルの行の長さを変更します.

```
fits.table("EVENT").col(0L).assign_default(NAN);
fits.table("EVENT").resize_rows(100);
```

---

## 12.9 Ascii Table HDU・Binary Table HDU の操作 (低レベル)

ここでは，Ascii Table HDU と Binary Table HDU を操作するための低レベル API を解説します．低レベル API では，ヘッダの TZERO$n$，TSCAL$n$ の値による変換処理ついてはユーザ自身で行なう必要があります．通常，ここで取り上げる API は必要ありませんが，高速化などのためには有用かもしれません．

### 12.9.1 table().col().data_array_cs()

**NAME**

table().col().data_array_cs() — データバッファ管理オブジェクトの参照

**SYNOPSIS**

```
const sli::mdarray &table( ... ).col( ... ).data_array_cs() const;
```

**DESCRIPTION**

fits_table_col クラスのテーブルデータは，SLLIB の mdarray クラスを使って管理しています．data_array_cs() メンバ関数は，ユーザが mdarray クラスを使って演算等を行ないたい場合に利用します．

mdarray クラスの詳細は，SLLIB のマニュアルを参照してください．

---

### 12.9.2 table().col().data_ptr()

**NAME**

table().col().data_ptr() — オブジェクト内データバッファのアドレス

**SYNOPSIS**

```
void *table( ... ).col( ... ).data_ptr();
```

**DESCRIPTION**

内部テーブルデータバッファのアドレスを返します．

返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，型やサイズが変更された場合は無効になります．

返されたアドレスは，現在のカラムの型に応じて，`fits::double_t *`，`fits::float_t *`，`fits::longlong_t *`，`fits::long_t *`，`fits::short_t *`，`fits::byte_t *` `fits::logical_t *` のいずれかの型にキャストして使います．

**RETURN VALUE**

内部テーブルデータバッファのアドレスを返します．

**EXAMPLES**

```
fits::double_t *tbl_data_ptr
```

```
        = (fits::double_t *)fits.table("EVENT").col(0L).data_ptr();
            :
            :
```

---

### 12.9.3 table().col().get_data()

**NAME**

table().col().get_data() — データを外部バッファへコピー

**SYNOPSIS**

```
ssize_t *table( ... ).col( ... )
                      .get_data( void *dest_buf, size_t buf_size ) const;
ssize_t *table( ... ).col( ... )
                      .get_data( long row_idx,
                                 void *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

カラムの生のデータを，row_idx で指定された行から最大で buf_size バイト，dest_buf へコピーします.

dest_buf で指定したアドレスは, 現在のカラムの型に応じて, fits::double_t *, fits::float_t *, fits::longlong_t *, fits::long_t *, fits::short_t *, fits::byte_t * fits::logical_t * のいずれかの型にキャストして使います.

**PARAMETER**

[O] dest_buf コピー先となるバッファ領域のアドレス
[I] buf_size dest_buf のバイトサイズ
[I] row_idx コピー元 (内部バッファ) の行インデックス
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : バッファ長が十分な場合にコピーできるバイト数.
負の値 (エラー) : 引数不正でコピーされなかった場合.

**EXAMPLES**

次のコードは，テーブル「EVENT」のカラム 0 のすべての内容を，ユーザのバッファにコピーします.

```
fits_table_col &col_ref = fits.table("EVENT").col(0L);
size_t buf_size = col_ref.elem_byte_length() * col_ref.length();
char *dest_buf = (char *)malloc(buf_size);
if ( dest_buf == NULL ) {
    エラー処理
}
col_ref.get_data(dest_buf, buf_size);
```

---

### 12.9.4 table().col().put_data()

**NAME**

table().col().put_data() — 外部バッファのデータを取り込む

**SYNOPSIS**

```
ssize_t *table( ... ).col( ... ).put_data( const void *src_buf, size_t buf_size );
ssize_t *table( ... ).col( ... )
                      .put_data( long row_idx, const void *src_buf, size_t buf_size );
```

**DESCRIPTION**

src_buf の生データを，row_idx で指定された行からオブジェクトの内部バッファへ最大で buf_size バイトコピーします.

**PARAMETER**

[I] src_buf コピー元となるバッファ領域のアドレス
[I] buf_size src_buf のバイトサイズ
[I] row_idx コピー先 (内部バッファ) の行インデックス
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 ： src_buf のバッファ長が十分な場合にコピーできるバイト数.
負の値 (エラー) ： 引数不正でコピーされなかった場合.

**EXAMPLES**

次のコードは，ユーザのバッファから，テーブル「EVENT」のカラム 0 のすべての内容を更新します.

```
fits_table_col &col_ref = fits.table("EVENT").col(0L);
size_t buf_size = col_ref.elem_byte_length() * col_ref.length();
char *data_buf = (char *)malloc(buf_size);
    :
    :
col_ref.put_data(data_buf, buf_size);
```

---

### 12.9.5 table().heap_ptr()

**NAME**

table().heap_ptr() — 可変長配列用のヒープバッファのアドレス

**SYNOPSIS**

```
void *table( ... ).heap_ptr();
```

**DESCRIPTION**

可変長配列用のヒープバッファの先頭アドレスを返します.

ヒープバッファ上のデータはビッグエンディアンで格納され，アライメントも保証されません. したがって，先頭以外のアドレスから値を読む場合，ヒープ上の必要な部分を別バッファにバ

イト単位でコピーし，別バッファでエンディアンを変換してから読まなければなりません (書き込む場合はその逆を行なう必要があります).

返される値はオブジェクト内部バッファのアドレスですから，オブジェクトが破棄されたり，サイズが変更された場合は無効になります.

一般的な用途には，table().get_heap() または，table().put_heap() の使用をお勧めします (§12.9.6, §12.9.7を参照).

---

### 12.9.6 table().get_heap()

**NAME**

table().get_heap() — 可変長配列用のヒープデータを外部バッファへコピー

**SYNOPSIS**

```
ssize_t *table( ... ).get_heap( void *dest_buf, size_t buf_size ) const;
ssize_t *table( ... ).get_heap( long offset,
                                void *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

可変長配列用のヒープバッファ上のデータを，offset で指定された位置から最大で buf_size バイト，dest_buf へコピーします.

ヒープバッファはビッグエンディアンでデータが格納されているため，dest_buf で指定したデータはエンディアンを変換して使います.

**PARAMETER**

[O] dest_buf コピー先となるバッファ領域のアドレス
[I] buf_size dest_buf のバイトサイズ
[I] offset コピー元での開始位置 (バイトオフセット)
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 ： バッファ長が十分な場合にコピーできるバイト数.
負の値 (エラー) ： 引数不正でコピーされなかった場合.

**EXAMPLES**

ソースパッケージに含まれる test/access_bte_heap.cc を参照してください.

---

### 12.9.7 table().put_heap()

**NAME**

table().put_heap() — 外部バッファのデータを可変長配列用のヒープ領域にコピー

**SYNOPSIS**

```
ssize_t *table( ... ).put_heap( const void *src_buf, size_t buf_size );
ssize_t *table( ... ).put_heap( long offset,
                                const void *src_buf, size_t buf_size );
```

**DESCRIPTION**

src_buf のデータを，オブジェクト内部のヒープバッファへ offset で指定された (ヒープの) バイト位置から最大で buf_size バイトコピーします.

.put_heap() を使う前に，src_buf で指定したデータはビッグデンディアンに変換しておく必要があります.

**PARAMETER**

[I] src_buf コピー元となるバッファ領域のアドレス
[I] buf_size src_buf のバイトサイズ
[I] offset コピー先 (内部ヒープバッファ) の開始位置 (バイトオフセット)
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 ： src_buf のバッファ長が十分な場合にコピーできるバイト数.
負の値 (エラー) ： 引数不正でコピーされなかった場合.

**EXAMPLES**

ソースパッケージに含まれる sample/create_vl_array.cc を参照してください.

---

### 12.9.8 table().resize_heap()

**NAME**

table().resize_heap() — ヒープ領域のサイズを変更

**SYNOPSIS**

```
fits_table &table( ... ).resize_heap( size_t sz );
```

**DESCRIPTION**

オブジェクト内部のヒープバッファの大きさを sz バイトに変更します.

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

ソースパッケージに含まれる sample/create_vl_array.cc を参照してください.

---

### 12.9.9 table().reverse_heap_endian()

**NAME**

table().reverse_heap_endian() — ヒープ領域の一部のエンディアンを反転

**SYNOPSIS**

```
fits_table &table( ... ).reverse_heap_endian( long offset,
                                              int type, long length );
```

**DESCRIPTION**

オブジェクト内部のヒープバッファのアドレス `offset` から始まるバイトデータを，データ型を `type` とする長さ `length` の配列とみなし，**必要に応じて**その部分のエンディアンを反転させます．

`type` に指定できる値は次のとおりです: `FITS::SHORT_T`, `FITS::LONG_T`, `FITS::LONGLONG_T`, `FITS::FLOAT_T`, `FITS::DOUBLE_T`, `FITS::COMPLEX_T`, `FITS::DOUBLECOMPLEX_T`.

エンディアンの反転が行なわれるのは，指定されたデータ型の計算機におけるエンディアンが little-endian の場合です．逆に，big-endian の計算機においては，このメンバ関数は何もしません．

**PARAMETER**

[I] `offset` ヒープ上のアドレス (0-indexed，バイト単位)
[I] `type` 要素のデータ型
[I] `length` 要素の個数
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します．

---

### 12.9.10 table().reserved_area_length()

**NAME**

table().reserved_area_length() — 予約領域のバイト長

**SYNOPSIS**

```
long long table( ... ).reserved_area_length() const;
```

**DESCRIPTION**

バイナリテーブル HDU において，Data Unit 中の予約領域 (Reserved Area) のバイトサイズを返します．

予約領域については，§3.5 をご覧ください．

---

### 12.9.11 table().resize_reserved_area()

**NAME**

table().resize_reserved_area() — 予約領域のバイト長を変更

**SYNOPSIS**

```
fits_table &table( ... ).resize_reserved_area( long long sz );
```

**DESCRIPTION**

バイナリテーブル HDU において，Data Unit 中の予約領域 (Reserved Area) の大きさを `sz` バイトに変更します．

予約領域については，§3.5 をご覧ください．

**RETURN VALUE**

当該 fits_table オブジェクトの参照を返します．

---

### 12.9.12 table().col().short_value()

**NAME**

table().col().short_value() — セルの生の値を整数値 (short 型) で返す

**SYNOPSIS**

```
short table( ... ).col( ... ).short_value( long row_index ) const;
short table( ... ).col( ... ).short_value( long row_index,
                          const char *elem_name, long repetition_idx = 0 ) const;
short table( ... ).col( ... ).short_value( long row_index,
                              long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を整数値 (short 型) で返します．カラムの型が FITS::SHORT_T (TFORM$n$ の指定に'I' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

実数型のカラムの場合は，最も近い整数値に丸めた値を返します．

論理型のカラムの場合は，値が'T' なら 1 を，それ以外の場合は 0 を返します．

文字列型のカラムの場合は，文字列を実数に変換し，最も近い整数値に丸めた値を返します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I] : 入力，[O] : 出力)

**RETURN VALUE**

セルの値を返します．

**EXAMPLES**

```
short value = fits.table("EVENT").col(0L).short_value(0);
        :
        :
```

---

### 12.9.13 table().col().long_value()

**NAME**

table().col().long_value() — セルの生の値を整数値 (long 型) で返す

**SYNOPSIS**

```
long table( ... ).col( ... ).long_value( long row_index ) const;
long table( ... ).col( ... ).long_value( long row_index,
                          const char *elem_name, long repetition_idx = 0 ) const;
long table( ... ).col( ... ).long_value( long row_index,
                             long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を整数値 (long 型) で返します．カラムの型が FITS::LONG_T (TFORM$n$ の指定に 'J' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

実数型のカラムの場合は，最も近い整数値に丸めた値を返します．

論理型のカラムの場合は，値が'T' なら 1 を，それ以外の場合は 0 を返します．

文字列型のカラムの場合は，文字列を実数に変換し，最も近い整数値に丸めた値を返します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します．

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください．

---

### 12.9.14 table().col().longlong_value()

**NAME**

table().col().longlong_value() — セルの生の値を整数値 (long long 型) で返す

**SYNOPSIS**

```
long long table( ... ).col( ... ).longlong_value( long row_index ) const;
long long table( ... ).col( ... ).longlong_value( long row_index,
                          const char *elem_name, long repetition_idx = 0 ) const;
long long table( ... ).col( ... ).longlong_value( long row_index,
                             long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を整数値 (long long 型) で返します．カラムの型が FITS::LONGLONG_T (TFORM$n$ の指定に'K' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

実数型のカラムの場合は，最も近い整数値に丸めた値を返します．

論理型のカラムの場合は，値が'T' なら 1 を，それ以外の場合は 0 を返します．

文字列型のカラムの場合は，文字列を実数に変換し，最も近い整数値に丸めた値を返します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します．

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください．

---

### 12.9.15 table().col().byte_value()

**NAME**

table().col().byte_value() — セルの生の値を整数値 (byte 型) で返す

**SYNOPSIS**

```
unsigned char table( ... ).col( ... ).byte_value( long row_index ) const;
unsigned char table( ... ).col( ... ).byte_value( long row_index,
                          const char *elem_name, long repetition_idx = 0 ) const;
unsigned char table( ... ).col( ... ).byte_value( long row_index,
                              long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を整数値 (byte 型) で返します．カラムの型が FITS::BYTE_T (TFORM$n$ の指定に'B' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

実数型のカラムの場合は，最も近い整数値に丸めた値を返します．

論理型のカラムの場合は，値が'T' なら 1 を，それ以外の場合は 0 を返します．

文字列型のカラムの場合は，文字列を実数に変換し，最も近い整数値に丸めた値を返します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します．

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください．

---

### 12.9.16 table().col().float_value()

**NAME**

table().col().float_value() — セルの生の値を実数値 (float 型) で返す

**SYNOPSIS**

```
float table( ... ).col( ... ).float_value( long row_index ) const;
float table( ... ).col( ... ).float_value( long row_index,
                          const char *elem_name, long repetition_idx = 0 ) const;
float table( ... ).col( ... ).float_value( long row_index,
                             long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を実数値 (float 型) で返します．カラムの型が FITS::FLOAT_T (TFORM$n$ の指定に 'E' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

論理型のカラムの場合は，値が'T' なら 1 を，それ以外の場合は 0 を返します．

文字列型のカラムの場合は，文字列を実数に変換した値を返します．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス |
| [I] | repetiton_index | 2 次元目のインデックス (1 次元目のインデックス) |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します.

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください.

---

### 12.9.17 table().col().double_value()

**NAME**

table().col().double_value() — セルの生の値を実数値 (double 型) で返す

**SYNOPSIS**

```
double table( ... ).col( ... ).double_value( long row_index ) const;
double table( ... ).col( ... ).double_value( long row_index,
                          const char *elem_name, long repetition_idx = 0 ) const;
double table( ... ).col( ... ).double_value( long row_index,
                              long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を実数値 (double 型) で返します．カラムの型が FITS::DOUBLE_T (TFORM$n$ の指定に'D' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです.

論理型のカラムの場合は，値が'T' なら 1 を，それ以外の場合は 0 を返します.

文字列型のカラムの場合は，文字列を実数に変換した値を返します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します.

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください.

---

### 12.9.18 table().col().bit_value()

**NAME**

table().col().bit_value() — セルの生の値を整数値 (bit 型) で返す

**SYNOPSIS**

```
long table( ... ).col( ... ).bit_value( long row_index ) const;
long table( ... ).col( ... ).bit_value( long row_index,
            const char *elem_name, long repetition_idx = 0, int nbit = 0 ) const;
long table( ... ).col( ... ).bit_value( long row_index,
                long elem_index, long repetition_idx = 0, int nbit = 1 ) const;
```

**DESCRIPTION**

セルの生の値を整数値 (bit 型) で返します. カラムの型が FITS::BIT_T (TFORM$n$ の指定に'X' を含む) の場合に最速でアクセスできます.

row_index で行を, elem_name(名前) または elem_index(インデックス) で要素を指定します. elem_name には, TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合, 1 次元目のインデックスを elem_index で, 2 次元目のインデックスを repetition_idx で指定できます.

引数 nbit で, 指定した要素から右方向何ビットを値として使うかを指定できます. 引数 nbit が 0 の場合, TELEM$n$ で指定されたビットフィールドの情報を利用します (ビットフィールドについては, §10.10をご覧ください).

実数型のカラムの場合は, 最も近い整数値に丸めた値を返します. ただし, これらのメンバ関数で返される値は, ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです.

論理型のカラムの場合は, 値が'T' なら 1 を, それ以外の場合は 0 を返します.

文字列型のカラムの場合は, 文字列を実数に変換し, 最も近い整数値に丸めた値を返します.

引数のインデックスは, 0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I] : 入力, [O] : 出力)

**RETURN VALUE**

セルの値を返します.

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください.

---

### 12.9.19 table().col().logical_value()

**NAME**

table().col().logical_value() — セルの生の値を論理値で返す

**SYNOPSIS**

```
int table( ... ).col( ... ).logical_value( long row_index ) const;
int table( ... ).col( ... ).logical_value( long row_index,
                           const char *elem_name, long repetition_idx = 0 ) const;
int table( ... ).col( ... ).logical_value( long row_index,
                               long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を論理値で返します．返り値は，'T'，'F'，'U' の 3 種類です．カラムの型が FITS::LOGICAL_T (TFORM$n$ の指定に'L' を含む) の場合に最速でアクセスできます．

論理型のカラムの場合，値が'T' なら'T' を，'F' なら'F' を，これら以外なら'U' を返します．

文字列型のカラムの場合は，文字列を実数に変換し，その実数を最も近い整数値に丸めた後，その整数値が 0 なら'F' を，0 でないなら'T' を返します．文字列が実数に変換できなかった場合，'T' か't' で始まる文字列なら'T' を，'F' か'f' で始まる文字列なら'F' を，そうでない場合は，'U' を返します．

実数型のカラムの場合は，最も近い整数値に丸めた後，その整数値が 0 なら'F' を，0 でないなら'T' を返します．整数型のカラムの場合，値が 0 なら'F' を，0 でないなら'T' を返します．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値を返します．

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください．

### 12.9.20 table().col().string_value()

**NAME**

table().col().string_value() — セルの生の値を文字列値で返す

**SYNOPSIS**

```
const char *table( ... ).col( ... ).string_value( long row_index ) const;
const char *table( ... ).col( ... ).string_value( long row_index,
                          const char *elem_name, long repetition_idx = 0 ) const;
const char *table( ... ).col( ... ).string_value( long row_index,
                               long elem_index, long repetition_idx = 0 ) const;
```

**DESCRIPTION**

セルの生の値を文字列値で返します．カラムの型が FITS::ASCII_T (TFORM$n$ の指定に'A' を含む) の場合に最速でアクセスできます．

文字列型のカラムの場合は，生の文字列を返します．

論理型のカラムの場合は，"T"，"F"，"U"のいずれかを返します．

整数型のカラムの場合は，libc の printf() 関数のフォーマット"%lld"に従って変換した文字列を返します．実数型のカラムの場合は，libc の printf() 関数のフォーマット"%.15G"に従って変換した文字列を返します．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

セルの値文字列のアドレスを返します．

**EXAMPLES**

§12.9.12の EXAMPLES を参照してください．

---

### 12.9.21 table().col().array_heap_offset()

**NAME**

table().col().array_heap_offset() — ヒープエリア上でのデータ位置を返す

**SYNOPSIS**

```
long table( ... ).col( ... ).array_heap_offset( long row_index,
                                                long elem_index = 0 ) const;
```

**DESCRIPTION**

可変長配列を使ったカラムにおいて，当該行で参照しているヒープエリア上のバイトデータの位置を返します.

1 行で複数の可変長配列を参照している場合には，elem_index を指定し，どの可変長配列にアクセスするかを指定します.

引数のインデックスは，0 から始まる数字です.

可変長配列の長さを調べるには，.array_length() を使います (§12.8.18).

**PARAMETER**

[I] row_index 行インデックス
[I] elem_index 複数の可変長配列がある場合，そのインデックス (省略可)
([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 : ヒープエリア上でのデータ位置 (先頭を 0 とし，バイト単位).
負の値 (エラー) : 引数が不正な場合や，カラムが可変長配列を使っていない場合など.

**EXAMPLES**

ソースパッケージに含まれる test/access_bte_heap.cc のコードを参照してください.

---

### 12.9.22 table().col().get_string_value()

**NAME**

table().col().get_string_value() — セルの生の値を文字列値で得る

**SYNOPSIS**

```
ssize_t table( ... ).col( ... ).get_string_value( long row_index,
                                        char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_string_value( long row_index,
                                        const char *elem_name,
                                        char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_string_value( long row_index,
                                    const char *elem_name, long repetition_idx,
                                    char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_string_value( long row_index,
                                        long elem_index,
                                        char *dest_buf, size_t buf_size ) const;
ssize_t table( ... ).col( ... ).get_string_value( long row_index,
                                        long elem_index, long repetition_idx,
                                        char *dest_buf, size_t buf_size ) const;
```

**DESCRIPTION**

セルの生の値を文字列値にして dest_buf に返します．dest_buf のバッファの容量 (バイト数) は buf_size で指定します．

文字列型のカラムの場合は，生の文字列を得ます．

論理型のカラムの場合は，TDISP$n$ の指定が無ければ，"T"，"F"，"U"のいずれかを得ます．

整数型のカラムの場合は，libc の printf() 関数のフォーマット"%lld"に従って変換した文字列を得ます．実数型のカラムの場合は，libc の printf() 関数のフォーマット"%.15G"に従って変換した文字列を得ます．ただし，これらのメンバ関数で返される値は，ヘッダの TZERO$n$ と TSCAL$n$ を**反映していない**ものです．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |
| [O] | dest_buf | 文字列データの取得領域アドレス |
| [I] | buf_size | dest_buf のバイトサイズ |

([I]：入力，[O]：出力)

**RETURN VALUE**

非負の値 ： バッファ長が十分な場合にコピーできる文字数 ('\0' は含まない)．
負の値 (エラー) ： 引数不正でコピーされなかった場合．

**EXCEPTION**

内部メモリ領域確保の操作に失敗した場合，SFITSIO に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

```
char buf[128];
fits.table("EVENT").col(0L).get_string_value( 0, buf, sizeof(buf) );
```

---

### 12.9.23 table().col().assign_short()

**NAME**

table().col().assign_short() — 整数値 (short 型) でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_short( short value,
                                                      long row_index );
fits_table_col &table( ... ).col( ... ).assign_short( short value,
```

```
                long row_index, const char *elem_name, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... ).assign_short( short value,
                long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

整数値 (short 型) でセルに値をそのまま代入します．カラムの型が FITS::SHORT_T (TFORM$n$ の指定に'I' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数では，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**.

論理型のカラムの場合は，value が 0 なら'F' を，そうでない場合は'T' を格納します.

文字列型のカラムの場合は，value を printf() のフォーマット"%hd"の形式で文字列に変換したものを格納します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，value 値のフォーマット変換に失敗した場合)，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

```
short value = 0;
fits.table("EVENT").col(0L).assign_short(value, 0);
```

---

### 12.9.24 table().col().assign_long()

**NAME**

table().col().assign_long() — 整数値 (long 型) でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_long( long value,
                                                     long row_index );
fits_table_col &table( ... ).col( ... ).assign_long( long value,
```

```
                long row_index, const char *elem_name, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... ).assign_long( long value,
                long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

整数値 (long 型) でセルに値をそのまま代入します．カラムの型が FITS::LONG_T (TFORM$n$ の指定に'J' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数では，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**.

論理型のカラムの場合は，value が 0 なら'F' を，そうでない場合は'T' を格納します.

文字列型のカラムの場合は，value を printf() のフォーマット"%ld"の形式で文字列に変換したものを格納します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，value 値のフォーマット変換に失敗した場合)，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください.

---

### 12.9.25 table().col().assign_longlong()

**NAME**

table().col().assign_longlong() — 整数値 (long long 型) でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_longlong( long long value,
                                                         long row_index );
fits_table_col &table( ... ).col( ... ).assign_longlong( long long value,
                long row_index, const char *elem_name, long repetition_idx = 0 );
```

```
fits_table_col &table( ... ).col( ... ).assign_longlong( long long value,
              long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

整数値 (long long 型) でセルに値をそのまま代入します．カラムの型が FITS::LONGLONG_T (TFORM$n$ の指定に'K' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数では，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**.

論理型のカラムの場合は，value が 0 なら'F' を，そうでない場合は'T' を格納します.

文字列型のカラムの場合は，value を printf() のフォーマット"%lld"の形式で文字列に変換したものを格納します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，value 値のフォーマット変換に失敗した場合)，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください.

---

### 12.9.26 table().col().assign_byte()

**NAME**

table().col().assign_byte() — 整数値 (byte 型) でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_byte( unsigned char value,
                                                     long row_index );
fits_table_col &table( ... ).col( ... ).assign_byte( unsigned char value,
              long row_index, const char *elem_name, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... ).assign_byte( unsigned char value,
              long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

整数値 (byte 型) でセルに値をそのまま代入します．カラムの型が FITS::BYTE_T (TFORM$n$ の指定に'B' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数では，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**.

論理型のカラムの場合は，value が 0 なら'F' を，そうでない場合は'T' を格納します.

文字列型のカラムの場合は，value を printf() のフォーマット"%hhu"の形式で文字列に変換したものを格納します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，value 値のフォーマット変換に失敗した場合)，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください.

---

### 12.9.27 table().col().assign_float()

**NAME**

table().col().assign_float() — 実数値 (float 型) でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_float( float value,
                                                      long row_index );
fits_table_col &table( ... ).col( ... ).assign_float( float value,
              long row_index, const char *elem_name, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... ).assign_float( float value,
              long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

実数値 (float 型) でセルに値をそのまま代入します．カラムの型が FITS::FLOAT_T (TFORM$n$ の

指定に'E' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数では，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**.

整数型のカラムの場合は，最も近い整数値に丸めた値を格納します.

論理型のカラムの場合は，value を最も近い整数値に丸めた値が 0 なら'F' を，そうでない場合は'T' を格納します.

文字列型のカラムの場合は，printf() のフォーマット"%.15G"の形式で文字列に変換したものを格納します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します. elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，value 値のフォーマット変換に失敗した場合), SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください.

---

### 12.9.28 table().col().assign_double()

**NAME**

table().col().assign_double() — 実数値 (double 型) でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_double( double value,
                                                       long row_index );
fits_table_col &table( ... ).col( ... ).assign_double( double value,
              long row_index, const char *elem_name, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... ).assign_double( double value,
              long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

実数値 (double 型) でセルに値をそのまま代入します．カラムの型が FITS::DOUBLE_T (TFORM$n$ の指定に'D' を含む) の場合に最速でアクセスできます．ただし，これらのメンバ関数では，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**.

整数型のカラムの場合は，最も近い整数値に丸めた値を格納します.

論理型のカラムの場合は，value を最も近い整数値に丸めた値が 0 なら'F' を，そうでない場合は'T' を格納します.

文字列型のカラムの場合は，printf() のフォーマット"%.15G"の形式で文字列に変換したものを格納します.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，value 値のフォーマット変換に失敗した場合)，SLLIB に由来する例外 (sli::err_rec 例外) が発生します.

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください.

---

### 12.9.29 table().col().assign_bit()

**NAME**

table().col().assign_bit() — 整数値 (bit 型) でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_bit( long value,
                                                    long row_index );
fits_table_col &table( ... ).col( ... ).assign_bit( long value,
    long row_index, const char *elem_name, long repetition_idx = 0, nbit = 0 );
fits_table_col &table( ... ).col( ... ).assign_bit( long value,
         long row_index, long elem_index, long repetition_idx = 0, nbit = 1  );
```

**DESCRIPTION**

整数値 (bit 型) でセルに値をそのまま代入します．カラムの型が FITS::BIT_T (TFORM$n$ の指定に'X' を含む) の場合に最速でアクセスできます．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数 nbit で，指定した要素から右方向何ビットにアクセスするかを指定できます．引数 nbit が 0 の場合，TELEM$n$ で指定されたビットフィールドの情報を利用します．(ビットフィールドについては，§10.10をご覧ください)．

整数型と実数型のカラムの場合にも使えますが，これらのメンバ関数では，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**．

論理型のカラムの場合は，value が 0 なら'F' を，そうでない場合は'T' を格納します．

文字列型のカラムの場合は，value を printf() のフォーマット"%ld"の形式で文字列に変換したものを格納します．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します．

**EXCEPTION**

内部バッファの操作に失敗した場合 (例えば，value 値のフォーマット変換に失敗した場合)，SLLIB に由来する例外 (sli::err_rec 例外) が発生します．

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください．

---

### 12.9.30 table().col().assign_logical()

**NAME**

table().col().assign_logical() — 論理値でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_logical( int value,
                                                        long row_index );
fits_table_col &table( ... ).col( ... ).assign_logical( int value,
```

```
                long row_index, const char *elem_name, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... ).assign_logical( int value,
                long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

論理値でセルに値をそのまま代入します．カラムの型が FITS::LOGICAL_T (TFORM$n$ の指定に 'L' を含む) の場合に最速でアクセスできます．

論理型のカラムの場合は，value が'T' の場合は'T' を，value が'F' の場合は'F' を，これら以外なら'\0' をセットします．

整数型や実数型のカラムの場合は，value が'T' の場合は 1 を，それ以外なら 0 をセットします．ただし，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**．

文字列型のカラムの場合は，value が'T' の場合は"T"を，value が'F' の場合は"F"を，これら以外なら"U" をセットします．

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます．

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます．

引数のインデックスは，0 から始まる数字です．

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します．

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください．

---

### 12.9.31 table().col().assign_string()

**NAME**

table().col().assign_string() — 文字列値でセルに値をそのまま代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_string( const char *value,
                                                       long row_index );
fits_table_col &table( ... ).col( ... ).assign_string( const char *value,
                long row_index, const char *elem_name, long repetition_idx = 0 );
fits_table_col &table( ... ).col( ... ).assign_string( const char *value,
                long row_index, long elem_index, long repetition_idx = 0 );
```

**DESCRIPTION**

文字列値でセルに値をそのまま代入します.

論理型のカラムの場合は，value が実数に変換できた場合はその値が 0 なら'F' を，そうでないなら'T' を格納します．value が実数に変換できない場合は，value が'T' か't' で始まる文字列なら'T' を，'F' か'f' で始まる文字列なら'F' を，そうでない場合は，'\0' を格納します.

実数型のカラムの場合は，value を実数に変換した値を格納します．整数型のカラムの場合は，value を実数に変換し，最も近い整数値に丸めた値を格納します．ただし，ヘッダの TZERO$n$ と TSCAL$n$ の値による**変換を行ないません**.

row_index で行を，elem_name(名前) または elem_index(インデックス) で要素を指定します．elem_name には，TELEM$n$ に存在する名前を指定できます.

TDIM$n$ が指定されている場合，1 次元目のインデックスを elem_index で，2 次元目のインデックスを repetition_idx で指定できます.

引数のインデックスは，0 から始まる数字です.

**PARAMETER**

| | | |
|---|---|---|
| [I] | value | 代入する値 |
| [I] | row_index | 行インデックス |
| [I] | elem_name | 要素名 |
| [I] | elem_index | 要素インデックス (TDIM$n$ の 1 次元目のインデックス) |
| [I] | repetiton_index | TDIM$n$ の 2 次元目のインデックス |

([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXAMPLES**

§12.9.23の EXAMPLES を参照してください.

---

### 12.9.32 table().col().assign_arrdesc()

**NAME**

table().col().assign_arrdesc() — 可変長配列の配列記述子を代入

**SYNOPSIS**

```
fits_table_col &table( ... ).col( ... ).assign_arrdesc( long length, long offset,
                                                         long row_index,
                                                         long elem_index = 0 );
```

**DESCRIPTION**

引数で指定された行 (row_index) の可変長配列の配列記述子 (配列長: length，バイト単位のアドレス: offset) をテーブル本体のセルに代入します.

当該カラムが 1 行あたり複数の可変長配列を扱っている場合は，elem_index でアクセスすべき配列を選択します.

引数のアドレス値とインデックスは，0 から始まる数字です.

**PARAMETER**

[I] `length` 配列の個数
[I] `offset` 参照すべきヒープ上のアドレス (0-indexed，バイト単位)
[I] `row_index` 行インデックス
[I] `elem_index` 複数の可変長配列がある場合，そのインデックス (省略可)
([I]：入力，[O]：出力)

**RETURN VALUE**

当該 fits_table_col オブジェクトの参照を返します.

**EXAMPLES**

ソースパッケージに含まれる `sample/create_vl_array.cc` を参照してください.

---

# 13 APPENDIX1: サンプルプログラム

SFITSIO を使うプログラマにとって参考になるサンプルプログラムが SFITSIO のソースパッケージに含まれていますので，それらを簡単に紹介します.

- `sample/read_and_write.cc`

  FITS ファイルの内容をすべて読んで，別のファイルへ書き出すだけの簡単なコード.

- `sample/create_image.cc`

  §2.4で紹介したコードと同様ですが，このコードでは符号なし 16-bit の画像を作ります.

- `sample/create_image_and_header.cc`

  2 つの Image HDU を持つ FITS を新規に作るコード．様々なメンバ関数を使ってヘッダの作成・編集を行ない，fits_image オブジェクトの内部バッファに直接アクセスして画像を作ります.

- `sample/create_bintable.cc`

  バイナリテーブルを新規に作るコード．NULL 値の扱いと行数を増やした場合のデフォルト値の扱いについて参考になるでしょう.

- `sample/create_asciitable.cc`

  アスキーテーブルを新規に作るコード.

- `sample/dump_table.cc`

  任意の HDU のアスキーテーブルまたはバイナリテーブルの内容を表示するコード．ただし，可変長配列には対応していません.

- `sample/create_vl_array.cc`

  低レベル API を使って可変長配列を持つバイナリテーブルを新規に作るコード．上級者向け.

- `sample_wcs/wcs_test.cc`

  WCSTools の libwcs と SFITSIO とを使い，ピクセル座標から実座標を取得し，表示するコードです.

- `tools/conv_bitpix.cc`

  Primary HDU の画像の BITPIX を変換するコード.

  `image().convert_type()` メンバ関数についての解説 (§12.6.13) もご覧ください.

- `tools/create_from_template.c`

  FITS テンプレート (テキストファイル) から新規の FITS ファイルを作成するコード．FITS テンプレートについては，`tools/template/`内のファイルをお試しいただけます.

  §8「テンプレート機能」もご覧ください.

- `tools/fill_header_comments.cc`

  SFITSIO 搭載の FITS ヘッダコメント辞書 (§14) を使って，FITS ファイルの空白のコメント部を可能な限り埋めるコード.

  `hdu(...).header_fill_blank_comments()` メンバ関数についての解説 (§12.4.38) もご覧ください.

- `tools/stat_pixels.cc`

  Primary HDU の画像の統計値を計算し，表示します．IRAF の imstat に似ています．

  `image(...).stat_pixels(...)` メンバ関数についての解説 (§12.6.42) もご覧ください．

- `tools/combine_images.cc`

  画像のコンバインを行ないます．

  コマンドの引数は IRAF の imcombine に似ていますが，用いられている手法は IDL 等でよく行なわれる「複数の画像データを三次元の配列に入れてからコンバインを行なう」というものです．

  `image(...).combine_layers(...)` メンバ関数についての解説 (§12.6.42) もご覧ください．

- `tools/hv.cc`

  低レベル API を使ってヘッダを高速に表示します．上級者向け．

  ローカルディスク上にある非圧縮の FITS の場合はシークを使って必要な部分だけを読み込むので非常に高速です．`EXTEND` が `F` の場合は Data Unit を読み取らないため，圧縮されている FITS でも (ネットワーク経由でも) 非常に高速です．

  §12.5.1からの解説もご覧ください．

- `tools/dataunit_md5.cc`

  低レベル API を使って Data Unit 部の MD5 を表示します．上級者向け．

  このプログラムは，Data Unit 部の同一性チェックに利用できます．

# 14 APPENDIX2: SFITSIO 搭載の FITS ヘッダのコメント辞書

ライブラリ内において，FITS ヘッダのキーワードに対するコメント辞書は連想配列で保持しています．下記コード中に存在するキーワードについては，メンバ関数によっては自動的にヘッダレコードにコメントが入ります．

```
/*
 * See http://heasarc.gsfc.nasa.gov/docs/fcg/standard_dict.html for FITS
 * standard keywords and comments.
 */
static asarray_tstring Fallback_comments(               /* associative array */
        /*           |MIN                                        MAX| */
        /* system keywords */
        "SIMPLE",   "conformity to FITS standard",
        "BITPIX",   "number of bits per data pixel",
        "NAXIS",    "number of data axes",
        "NAXIS#",   "length of data axis #",
        "EXTEND",   "possibility of presence of extensions",
        "XTENSION", "type of extension",
        "PCOUNT",   "number of parameters per group",
        "GCOUNT",   "number of groups",
        "EXTNAME",  "name of this HDU",
        "EXTVER",   "version of the extension",
        "EXTLEVEL", "hierarchical level of the extension",
        /*           |MIN                                        MAX| */
        /* standard of JAXA data center */
        "FMTTYPE",  "type of format in FITS file",
        "FTYPEVER", "version of FMTTYPE definition",
        "FMTVER",   "version of FMTTYPE definition",
        /* standard keywords */
        "DATE",     "date of file creation",
        "ORIGIN",   "organization responsible for the data",
        "AUTHOR",   "author of the data",
        "REFERENC", "bibliographic reference",
        "DATE-OBS", "date of the observation",
        "TELESCOP", "telescope or mission name",
        "INSTRUME", "instrument name",
        "DETECTOR", "detector name",
        "OBSERVER", "observer who acquired the data",
        "OBJECT",   "name of observed object",
        /*           |MIN                                        MAX| */
        "EPOCH",    "equinox of celestial coordinate system",
        "EQUINOX",  "equinox of celestial coordinate system",
        "TIMESYS",  "explicit time scale specification",
        /* other general keywords */
        "OBSERVAT", "observatory name",
        "CREATOR",  "data generator program",
        "PIPELINE", "data processing pipeline name",
        "FILENAME", "file name",
        "PROPOSAL", "proposal ID",
        "BAND",     "band name",
        "MJD",      "modified Julian date",
        "AIRMASS",  "air mass",
        "EXPTIME",  "exposure time",
        "WEATHER",  "weather condition",
        /* general keyword of JAXA data center */
```

```
        "CNTTYPE",  "type of data content",
        "CNTVER",   "version of data content",
        "CHECKSUM", "HDU checksum",
        "DATASUM",  "data unit checksum",
        /*          |MIN                                    MAX| */
        NULL
);
static asarray_tstring Image_comments(                  /* associative array */
        /*          |MIN                                    MAX| */
        "BZERO",    "zero point in scaling equation",
        "BSCALE",   "linear factor in scaling equation",
        "BLANK",    "value used for undefined pixels",
        "BUNIT",    "physical unit of the pixel values",
        "DATAMIN",  "minimum data value",
        "DATAMAX",  "maximum data value",
        /* WCS */
        "WCSAXES?", "number of axes for WCS",
        "CRVAL#?",  "world coordinate at reference point",
        "CRPIX#?",  "pixel coordinate at reference point",
        "CDELT#?",  "world coordinate increment at reference point",
        "CROTA#",   "coordinate system rotation angle",
        "CTYPE#?",  "type of celestial system and projection system",
        /*          |MIN                                    MAX| */
        "CUNIT#?",  "units of the coordinates along axis",
        "PC#_#?",   "matrix of rotation (#,#)",
        "CD#_#?",   "matrix of rotation and scale (#,#)",
        "WCSNAME?", "name of WCS",
        "LONPOLE?", "native longitude of celestial pole",
        "LATPOLE?", "native latitude of celestial pole",
        "EQUINOX?", "equinox of celestial coordinate system",
        "MJD-OBS",  "modified Julian date of observation",
        "CNAME#?",  "description of CTYPE definition",
        "RADESYS?", "default coordinate system",
        /*          |MIN                                    MAX| */
        NULL
);
static asarray_tstring Binary_table_comments(           /* associative array */
        /*          |MIN                                    MAX| */
        "BITPIX",   "number of bits per data element",
        "NAXIS1",   "width of table in bytes",
        "NAXIS2",   "number of rows in table",
        "PCOUNT",   "length of reserved area and heap",
        "TFIELDS",  "number of fields in each row",
        "TXFLDKWD", "extended field keywords",                  /* JAXA ext. */
        "TTYPE#",   "field name",
        "TALAS#",   "aliases of field name",                    /* JAXA ext. */
        "TELEM#",   "element names",                            /* JAXA ext. */
        "TUNIT#",   "physical unit",
        "TDISP#",   "display format",
        "TFORM#",   "data format",
        "TDIM#",    "dimensionality of the array",
        "TZERO#",   "zero point in scaling equation",
        "TSCAL#",   "linear factor in scaling equation",
        "TNULL#",   "value used for undefined cells",
        "THEAP",    "byte offset to heap area",
        /* CFITSIO ext. */
        "TLMIN#",   "minimum value legally allowed",
```

```
        "TLMAX#",   "maximum value legally allowed",
        "TDMIN#",   "minimum data value",
        "TDMAX#",   "maximum data value",
        /*           |MIN                                        MAX| */
        NULL
);
static asarray_tstring Ascii_table_comments(            /* associative array */
        /*           |MIN                                        MAX| */
        "BITPIX",   "number of bits per data element",
        "NAXIS1",   "width of table in bytes",
        "NAXIS2",   "number of rows in table",
        "TFIELDS",  "number of fields in each row",
        "TXFLDKWD", "extended field keywords",                  /* JAXA ext. */
        "TTYPE#",   "field name",
        "TALAS#",   "aliases of field name",                    /* JAXA ext. */
        "TUNIT#",   "physical unit",
        "TFORM#",   "display format",
        "TBCOL#",   "starting position in bytes",
        "TZERO#",   "zero point in scaling equation",
        "TSCAL#",   "linear factor in scaling equation",
        "TNULL#",   "value used for undefined cells",
        "TLMIN#",   "minimum value legally allowed",
        "TLMAX#",   "maximum value legally allowed",
        "TDMIN#",   "minimum data value",
        "TDMAX#",   "maximum data value",
        /*           |MIN                                        MAX| */
        NULL
);
```

# 15 APPENDIX3: 便利な TSTRING クラスの使い方

SFITSIO の内部で使われている「tstring クラス」は，C 言語の使い勝手を残しつつスクリプト言語のような手軽さで文字列処理が行なえるクラスです[43]．SFITSIO を導入されている場合は，

```
#include <sli/tstring.h>
using namespace sli;
```

と書けば，すぐに使えます．

## オブジェクトの生成と文字列の代入・表示

```
    tstring my_string;
    my_string = "I am a SFITSIO user!";
```

以上でオブジェクトの生成と，生成したオブジェクトへの文字列の代入が完了しました．オブジェクト内部で，文字列用のバッファを自動管理しているので，ユーザは代入や編集時にバッファの大きさを気にする必要はありません．また，バッファ領域の開放は，スコープから抜けた時に自動的に行なわれますから，メモリリークの心配もありません．

.printf() で代入する事もできます．この場合も，バッファ領域を気にする必要はありません．

```
    my_string.printf("Today is %d/%d/%d", y, m, d);
```

次は，my_string の内容を printf() で表示してみます．

```
    printf("my_string = %s\n", my_string.cstr());
```

printf() 関数に与える場合のように，文字列のアドレスが必要な場合は，.cstr() を使います．

## 文字列の編集

```
    my_string = "I am ";                /* 代入 */
    my_string += "a SFITSIO ";          /* 追加 */
    my_string.append("user!");          /* 追加 */
```

既存のものに文字列を追加したい場合，上記の例のように「+=」あるいは.append() を使います．

今度は，上記文字列の "SFITSIO" の前に "super " を挿入してみます．

```
    my_string.insert(7,"super ");
```

これで，"I am a super SFITSIO user!" のできあがりです．

この他にも，置換を行なう.replace() や削除を行なう.erase()，Perl と同じ機能の.chop() や .chomp()，両端の空白を消去する.trim()，大文字･小文字への変換を行なう toupper()，tolower() など多数のメンバ関数が用意されています．

## 文字列の検索・正規表現の利用

内部文字列に 1 文字ずつアクセスしたい場合も，正規表現で検索・置換したい場合も，用意されているメンバ関数を使えば非常に簡単です．正規表現は，POSIX 拡張正規表現が利用可能です．

次のコードでは，my_string の内容を 1 文字ずつ表示しています．

[43] C++標準ライブラリの string を使うのも 1 つの方法ですが，tstring クラスの方が C 言語的な使い方ができるメンバ関数が豊富に用意されているので，C 言語ユーザの方はこちらの方がずっと敷居が低く，使い勝手が良いはずです．

```
size_t i;
for ( i=0 ; i < my_string.length() ; i++ ) {
    printf("ch[%d] = %c\n", i, my_string.cchr(i));
}
```

文字列の長さを得るには，.length() を使います．これは SFITSIO と同じですね．

「[]」を使って一文字ずつ読み書きする事もできます．次の例では，空白をアンダーバーに置換しています．

```
size_t i;
for ( i=0 ; i < my_string.length() ; i++ ) {
    if ( my_string[i] == ' ' ) my_string[i] = '_';
}
```

正規表現を使って，検索をしてみましょう．

```
ssize_t pos;
size_t len;
pos = my_string.regmatch("[A-Z]+", &len);
```

パターンにマッチした部分の位置を pos に，長さを len に得ます．

最後に，正規表現を使って置換する例を紹介します．

```
tstring my_string = "TSTRING  is     very    easy !";
my_string.regreplace("[ ]+", " ", true);
```

1 文字以上の空白文字をすべて，1 文字の空白に置き換えています．

## 公式マニュアル

PDF のマニュアルがあります．詳細は，こちらをご覧ください．

http://www.ir.isas.jaxa.jp/~cyamauch/sli/sllib.pdf

# 16 APPENDIX4: 便利な DIGESTSTREAMIO クラスの使い方

SFITSIO の.read_stream()，.write_stream() はネットワークや圧縮ファイルに対応していますが，これは digeststreamio クラスで実現しています．digeststreamio クラスを使うと，libc の fopen() 関数，fgets() 関数，fgets() 関数などを使う場合と同じようにコードを書くだけで，ネットワークへの接続，圧縮ストリームの圧縮・伸長を行なう事ができます．圧縮形式は，gzip と bzip2 に対応しています．

SFITSIO を導入されている場合は，

```
#include <sli/digeststreamio.h>
using namespace sli;
```

と書けば，すぐに使えます．

## ファイルのオープンとクローズ

```
    int status;
    digeststreamio dsio;
    status = dsio.open("r", "http://www.jaxa.jp/");
```

ファイルのオープンは，.open() を使います．第一引数に"r"(読み込み) か"w"(書き込み) を指定し，第二引数にパスを指定します．パスは，http://...，ftp://...，file://... が指定できます．file://は省略可能です．open() メンバ関数は，エラーの場合は負の値を返します．

ファイルのクローズは.close() を使います．

```
    dsio.close();
```

## 読み込み

libc の fgets() に相当するのは getstr() メンバ関数で，fgetc() に相当するのは getchr() メンバ関数です．digeststreamio のメンバ関数名は，character を示す場合は chr，string を示す場合は str を使っています．

次のように使います．

```
    char buf[256];
    while ( dsio.getstr(buf, 256) != NULL ) {
        printf("%s", buf);
    }
```

```
    int ch;
    while ( (ch = dsio.getchr()) != EOF ) {
        printf("%c", ch);
    }
```

.getline() を使うと，1 行ずつ改行まで読み取る事ができます．

```
    const char *ptr;
    while ( (ptr = dsio.getline()) != NULL ) {
        printf("%s", ptr);
    }
```

### 書き込み

文字列を書き込む場合は，.putstr() を使います.

```
    dsio.putstr(buf);
```

.printf() を使う事もできます.

```
    dsio.printf("Today is %d/%d/%d\n", y, m, d);
```

## STDSTREAMIO クラス

tstring クラスや digeststreamio クラスを提供している SLLIB は，libc のほとんどの機能を提供しています．日常的に使う printf() 関数なども SLLIB のクラスを使って書けば，コードをオブジェクト指向で統一的に書く事ができます.

例えば，libc の printf() や putchar() は，stdstreamio クラスのメンバ関数で置き換える事ができます．次の例は，http://www.jaxa.jp/ の HTML を表示するコードです.

```
#include <sli/digestststreamio.h>
#include <sli/stdststreamio.h>
#include <sli/tstring.h>
using namespace sli;

int main()
{
    tstring buf;
    digeststreamio dsin;
    stdstreamio sio;

    dsin.openf("r", "%s:/%s", "http", "/www.jaxa.jp/");
    while ( (buf = dsin.getline()) != NULL ) {
        sio.printf("%s", buf);
    }
    dsin.close();
    return 0;
}
```

.openf() を使うと，printf() 関数と同様に可変長引数でパスを指定できます．このように，SLLIB のクラスには関数名の最後に「f」がついたメンバ関数が多く用意されており，それらでは可変長引数が指定できるので短くコードを記述する事ができます.

stdstreamio クラスの親クラスは digeststreamio クラスのそれと同じなので，digeststreamio クラスの場合と全く同じメンバ関数が使えます.

## 公式マニュアル

PDF のマニュアルがあります．詳細は，こちらをご覧ください.

http://www.ir.isas.jaxa.jp/~cyamauch/sli/sllib.pdf