Technical documentation

libfmutil User Manual

29. November 2012

Norwegian
Meteorological Institute
met.no

Revision history

Version	Date	Comment	Responsible
Version 1.0	02.05.2007	First documentation.	Øystein Godøy
Version 1.01	03.07.2007	Added fmMalloc and ymdhm2fm1970 and warning messages concerning time zones on Red Hat Enterprise and Fedora Core computers.	Thomas Lavergne, Øystein Godøy
Version 1.02	05.10.2007	Added description of fmtouch, fmfileexist and fmfindfile	Øystein Godøy
Version 1.03	03.12.2007	Added documentation for new `printf`-like formats in `fmlogmsg()` and `fmerrmsg()`.	Thomas Lavergne
Version 1.04	19.12.2007	Added documentation for the new `fmstrsplit()` routine and for interfaces to `isodatetime` format.	Thomas Lavergne
Version 1.05	22.02.2008	Added `fmselalg`, and some minor additions in documentation elswhere.	Øystein Godøy
Version 1.06	11.09.2008	Added `fmbasename` and `fmstrjoin`.	Thomas Lavergne
Version 1.07	24.09.2008	Added `fmstring2fmtime` routine	Thomas Lavergne
Version 1.08	14.01.2009	Added `Cfepoch` (`YYYY-MM-DD hh:mm:ss`) datestring format	Thomas Lavergne
Version 1.09	15.04.2010	Added `correct_time_from_unit()`	Thomas Lavergne
Version 1.10	19.12.2011	Added grep-like functionality to `fmstrings` and modified `fmimage` in accordance with introduction of a HLHDF reader in `libfmio`.	Øystein Godøy
Version 1.11	29.11.2012	Added DOY format for `fmstring2fmtime.` Introduce `check_fmtime`. Start on a test suite.	Thomas Lavergne
Version 1.12	29.11.2012	Change default behaviour in fmtime routines: now check converted date string and barks if it is invalid.	Thomas Lavergne

Table of Contents

1 Introduction

1.1 What

`libfmutil` is a C library containing various functions useful for handling remote sensing data. It is specifically developed for the needs at the Norwegian Meteorological Institute, but might be useful to other as well.

1.2 Why

`libfmutil` have been developed to help handling remote sensing data although it also contain utility functions useful for general software development.

1.3 Where

`libfmutil` have been developed for the following purposes:

- Research and development at the Norwegian Meteorological Institute
- Operational processing of remote sensing data at the Norwegian Meteorological Institute

1.4 When

`libfmutil` builds on an earlier software library `libsatimg`. However `libsatimg` contained both utility functions and file format specifications. The latter complicated the building and use of the library as some of the file formats were defined by commercial companies delivering software to the Norwegian Meteorological Institute. This also complicated use of the library in projects involving external partners. Thus these file format parts have been isolated within a specific library `libfmio` and the commercial file formats have been excluded.

1.5 Who

`libfmutil` have been developed by scientists employed by the Norwegian Meteorological Institute. However, the library is free to be used by anyone that would find it useful. It is distributed using GNU General Public License when it is built without PROJ.4 support. PROJ.4 is distributed using a MIT license. See the PROJ.4 homepage for details on the PROJ.4 license.

The following persons have been involved in the software development:

Name	Function	Affiliation
Øystein Godøy	Coordinator, Software development	Norwegian Meteorological Institute (http://www.met.no/)
Steinar Eastwood	Software development	Norwegian Meteorological Institute (http://www.met.no/)
Thomas Lavergne	Software developer	Norwegian Meteorological Institute (http://www.met.no/)

2 Compilation and implementation

2.1 Requirements

`libfmutil` have some functionality that relies on the PROJ.4 library. However, the library can be built without PROJ.4. The functionality requiring PROJ.4 will then be disabled.

`libfmutil` have been developed on a Linux system running Fedora Core 3 and Fedora Core 5. Although being developed for portability, this has not been thoroughly tested yet.

2.2 Compilation

The following steps are required to build the software:

1. Unpack the library (distributed as a gzipped tarball) using `tar xvzf libfmutil.tgz`
2. Run `cvs export -r HEAD -d commonM4cros metnoBuildCommon/m4cros`
3. Run `autoreconf --install --verbose`
4. execute `configure`. The `–help` option of `configure` will list options. If e.g. PROJ.4 support is not needed, use `–without-proj`. If reconfiguration of the system is required, remember to run `make distclean`.
5. `make`
6. `make check`
7. `make install`

2.3 Test

A test suite was introduced for securing updates in the libraries did not break expected behaviour. The suite is enabled by running `make check` either from top-level, or from inside the `testsuite` directory.

3 Library variables and structures

3.1 Definitions

```
typedef enum {FMFALSE, FMTRUE} fmboolean, fmbool;
#define fmPI 3.141592654
#define DEG_TO_RAD 0.0174532925199432958
#define RAD_TO_DEG 57.29577951308232
#define FMREARTH 6371.
```

3.2 Return codes

```
typedef enum {
    FM_OK,
    FM_SYNTAX_ERR,
```

```
    FM_IO_ERR,
    FM_MEMALL_ERR,
    FM_MEMFREE_ERR
} fmreturncode;
```

3.3 Angles

```
typedef struct {
    fmgeopos *gpos;
    int npoints;
} fmsubtrack;


typedef struct {
    fmucsref ref;
    fmgeopos ll;
    fmindex ind;
    fmtime date;
    fmsubtrack subtrack;
    fmprojspec myproj;
} fmangreq;


typedef struct {
    fmgeopos st1;
    fmgeopos st2;
    fmgeopos tp;
} fmtriangpoints;


typedef struct {
    double soz; /* solar zenith angle */
    double saz; /* satellite zenith angle */
    double raz; /* relative azimuth angle */
    double aza; /* satellite azimuth angle */
    double azu; /* solar azimuth angle */
} fmangles;
```

3.4 Coordinate systems

```
typedef enum {MI,MEOS} fmprojspec;


typedef enum {NR,NS,AT,GR} fmtiles;


typedef struct {
    int row;
    int col;
} fmindex; /* 2D image index */


typedef struct { /* see fmcoord.c for details */
    double northings;
    double eastings;
```

```
} fmucspos; /* User Coordinate System, origo at North Pole */


typedef struct {
    double lat;
    double lon;
} fmgeopos; /* Geographical latitude, longitude */


typedef struct { /* see fmcoord.c for definitions */
    double Ax; /* horizontal pixel size in km */
    double Ay; /* vertical pixel size in km */
    double Bx; /* eastings of upper left corner of image in km */
    double By; /* northings of upper left corner of image in km */
    int iw; /* image size in horizontal */
    int ih; /* image size in vertical */
} fmucsref;


typedef struct {
    float truelat; /* True latitude for polar stereographic
projection */
    float gridrot; /* grid rotation compare to Greenwich meridian */
    float jnp; /* position of the North Pole, x-coordinate */
    float inp; /* position of the North Pole, y-coordinate */
    float ngridp; /* number of grid points between Equator and NP */
    float dummy; /* only used for rotated grids, not handled herein
yet */
} fmdianapos;


static char *meosproj[] = {
    "proj=stere",
    "lat_0=90.0",
    "lon_0=0.0",
    "lat_ts=60.0",
    "units=km",
    "a=6378144.0",
    "b=6356759.0"
};


static char *miproj[] = {
    "proj=stere",
    "lat_0=90.0",
    "lon_0=0.0",
    "lat_ts=60.0",
    "units=km",
    "a=6371000.0",
    "b=6371000.0"
};
```

3.5 Image

```
typedef struct {
    float gain;
```

```
    float intercept;
    char description[50];
} fmslope;


typedef struct {
    fmslope *slope;
    int nslopes;
} fmscale;


typedef enum fmdatatype {
    FMUCHAR=0,
    FMCHAR,
    FMUSHORT,
    FMSHORT,
    FMUINT,
    FMINT,
    FMFLOAT,
    FMDOUBLE
} fmdatatype;

typedef struct {
    char platform_name[FMIMAGESTR25];  /* platform name */
    char satellite_name[FMIMAGESTR25];  /* satellite name */
    char sensor_name[FMIMAGESTR25];  /* sensor name */
    char area_description[FMIMAGESTR25];  /* area name */
    char processing_description[FMIMAGESTR50];  /* process name etc
*/
    char product_description[FMIMAGESTR1024]; /* free text
description */
    char distribution_constraints[FMSTRING128];
    char use_constraints[FMSTRING128];
    int orbit_no;
    fmbool map_projected;
    fmbool ucs_positioned;
    fmbool subtrack_added;
    int xsize;
    int ysize;
    float nominal_grid_resolution_x;
    float nominal_grid_resolution_y;
    fmtime valid_time; /* Reception or processing time */
    fmtime timespan[2]; /* Start and end time of data */
    fmucsref ucs; /* User Coordinate System, ucs_positioned */
    fmsubtrack subtrack; /* subsatellite track in latlon, if
subtrack_added */
    int layers; /* no. of layers */
    char **layer_id; /* layer names, not in use yet!! */
    char projstring[FMIMAGESTR1024]; /* PROJ.4 specification used */
} fmheader;

typedef struct {
    unsigned char **bytearray; /* image data */
    unsigned short **ushortarray; /* image data */
    int **intarray; /* image data */
```

```
    float **floatarray; /* image data */
    double **doublearray; /* image data */
    char *description;
    char *unit; /* spell it out in SI */
    fmdatatype dtype;
    int missingdatavalue; /* Must be interpreted according to data
field */
    int nodatavalue; /* Must be interpreted according to data field
*/
    fmbool packed; /* real values packed as int or similar? */
    fmscale scalefactor; /* intercept and slope for packed
information */
    fmbool palette; /* true if palette / classed image */
    int number_of_classes;
    char **class_names;
} fmdatafield;

typedef struct {
    fmheader h;
    fmdatafield *d;
} fmdataset;
```

3.6 Time

3.6.1 fmtime

```
typedef struct {
    int     fm_sec;    /* seconds */
    int     fm_min;    /* minutes */
    int     fm_hour;   /* hours */
    int     fm_mday;   /* day of the month */
    int     fm_mon;    /* month */
    int     fm_year;   /* year */
    int     fm_wday;   /* day of the week */
    int     fm_yday;   /* day in the year */
} fmtime;
```

An important notice concerning this function: The UK was on permanent British Summer Time (BST) during the Second World War from February 1940 until October 1945 and again from February 1968 until October 1971 (see http://wwp.greenwichmeantime.com/info/bst2.htm). When using `libfmutil` on computers running British Summer Time `fmtime` will give wrong results unless you specifically sets the TZ variable (on UNIX/Linux machines). Use (for Bash) `export TZ=GMT` to be safe. The software have been checked for CET, GMT, and various US time zones. It will fail for Singapore and Alaska as well as for the BST. If you are not running CET or GMT, you should check your time zone carefully and use GMT if uncertain.

3.6.2 fmsec1970

```
typedef long fmsec1970; /* unix time */
```

4 Library functions

4.1 Log functionality

4.1.1 fmlogmsg

```
void fmlogmsg(char *where, char *what,...);
```

Create a log message at standard out. `printf`-like format strings are allowed if standard header `stdarg.h` is available. Otherwise, the function prototype defaults to `void fmlogmsg(char *where, char *what);`.

Example:

```
fmlogmsg(“myprog”,”Library %s: the answer is %d.”,”libfmutil”,42);
```

4.1.2 fmerrmsg

```
void fmerrmsg(char *where, char *what,...);
```

Create an error message at standard error. `printf`-like format strings are allowed. See also `fmlogmsg`.

4.2 Memory management

Some of the functions described below are yet not fully tested!

4.2.1 fmalloc_char_vector

```
int fmalloc_char_vector(char **value, int length);
```

Dynamically allocate a character vector. Use `fmfree_char_vector` to release it.

4.2.2 fmalloc_ushort_vector

```
int fmalloc_ushort_vector(unsigned short **value, int length);
```

Dynamically allocate a unsigned short vector. Use `fmfree_ushort_vector` to release it.

4.2.3 fmalloc_int_vector

```
int fmalloc_int_vector(int **value, int length);
```

Dynamically allocate a integer vector. Use `fmfree_int_vector` to release it.

4.2.4 fmalloc_float_vector

```
int fmalloc_float_vector(float **value, int length);
```

Dynamically allocate a float vector. Use `fmfree_float_vector` to release it.

4.2.5 fmalloc_double_vector

```
int fmalloc_double_vector(double **value, int length);
```

Dynamically allocate a double vector. Use `fmfree_double_vector` to release it.

4.2.6 fmalloc_byte_2d

```
int fmalloc_byte_2d(char ***value, int rows, int columns);
```

Dynamically allocate a 2 dimensional array of character. Use `fmfree_byte_2d` to release it.

4.2.7 fmalloc_ubyte_2d

```
int fmalloc_ubyte_2d(unsigned char ***value, int rows, int columns);
```

Dynamically allocate a 2 dimensional array of unsigned character. Use `fmfree_ubyte_2d` to release it.

4.2.8 fmalloc_ushort_2d

```
int fmalloc_ushort_2d(unsigned short ***value, int rows, int columns);
```

Dynamically allocate a 2 dimensional array of unsigned short. Use `fmfree_ushort_2d` to release it.

4.2.9 fmalloc_int_2d

```
int fmalloc_int_2d(int ***value, int rows, int columns);
```

Dynamically allocate a 2 dimensional array of integer. Use `fmfree_int_2d` to release it.

4.2.10 fmalloc_float_2d

```
int fmalloc_float_2d(float ***value, int rows, int columns);
```

Dynamically allocate a 2 dimensional array of float. Use `fmfree_float_2d` to release it.

4.2.11 fmalloc_double_2d

```
int fmalloc_double_2d(double ***value, int rows, int columns);
```

Dynamically allocate a 2 dimensional array of double. Use `fmfree_double_2d` to release it.

4.2.12 fmfree_char_vector

```
int fmfree_char_vector(char *value);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.13 fmfree_ushort_vector

```
int fmfree_ushort_vector(unsigned short *value);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.14 fmfree_int_vector

```
int fmfree_int_vector(int *value);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.15 fmfree_float_vector

```
int fmfree_float_vector(float *value);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.16 fmfree_double_vector

```
int fmfree_double_vector(double *value);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.17 fmfree_byte_2d

```
int fmfree_byte_2d(char **value, int rows);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.18 fmfree_ubyte_2d

```
int fmfree_ubyte_2d(unsigned char **value, int rows);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.19 fmfree_ushort_2d

```
int fmfree_ushort_2d(unsigned short **value, int rows);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.20 fmfree_int_2d

```
int fmfree_int_2d(int **value, int rows);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.21 fmfree_float_2d

```
int fmfree_float_2d(float **value, int rows);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.22 fmfree_double_2d

```
int fmfree_double_2d(double **value, int rows);
```

Free dynamically allocated memory. See corresponding allocation functions.

4.2.23 fmivec

```
long fmivec(long x, long y, unsigned long nx);
```

Returns a pointer into the datavector containing the matrix with image data. This funtion is used when a 2 dimensional matrix is simulated using a 1 dimensional matrix.

4.2.24 fmijmap

```
void fmijmap(long elem, unsigned long nx, long *x, long *y);
```

Sister routine to `fmivec`. Returns two pointers (line and column index) into the matrix of

image data corresponding to element `elem` in the datavector. This funtion is used when a 2 dimensional matrix is simulated using a 1 dimensional matrix.

4.2.25 fmMalloc

```
void *fmMalloc(size_t size);
```

Wrapper for the standard `malloc` routine. It performs a test on the pointer allocated and terminates the program if necessary.

4.3 File handling functions

4.3.1 fmfileexist

```
int fmfileexist(char *filename);
```

Checks whether a specific file exist or not. Returns `FM_OK` if it exist and `FM_IO_ERR` if it does not exist.

4.3.2 fmtouch

```
int fmtouch(char *filename, time_t newtime);
```

Touches the specified file. Use `mktime` as input time if touching using the current timestamp is required.

4.3.3 fmfindfile

```
char *fmfindfile(char *path, char *datestr, char *timestr, char *areastr);
```

Checks the directory specified by `path` for filenames that contain the 3 strings defined by `datestr`, `timestr` and `areastr`.

4.4 Image containers

4.4.1 init_fmimage

```
int init_fmimage(fmimage *d);
```

To initialize the `fmimage` structure containing both image data and header information.

4.4.2 free_fmimage

```
int free_fmimage(fmimage *d);
```

To free dynamically allocated memory within the `fmimage` structure.

4.4.3 allocate_fmdatafield

```
int allocate_fmdatafield(fmdatafield **d, int layers);
```

To allocate the data container within `fmdataset`.

4.4.4 unpack_fmimage_ushort - REMOVED

~~`float unpack_fmimage_ushort(unsigned short val, fmscale scale, int`~~

~~use);~~

~~Unpacks floating point values from unsigned short to float. If e.g. AVHRR bi-directional reflectance or brightness temperatures are stored as unsigned short or integer values by the process generating the files, these values can be unpacked using this function. Typically brightness-temperatures are returned as Kelvin and bi-directional reflectance as percentage (0-100), but this is determined by the process encoding the data (e.g. KSPT MEOS, PolsatProc, etc.).~~

4.4.5 unpack_fmimage_int - REMOVED

~~float unpack_fmimage_int(int val, fmscale scale, int use);~~

~~Unpacks floating point values from unsigned short to float. If e.g. AVHRR bi-directional reflectance or brightness temperatures are stored as unsigned short or integer values by the process generating the files, these values can be unpacked using this function. Typically brightness-temperatures are returned as Kelvin and bi-directional reflectance as percentage (0-100), but this is determined by the process encoding the data (e.g. KSPT MEOS, PolsatProc, etc.).~~

4.4.6 unpack_fmdatafield_byte

```
float unpack_fmdatafield_byte(char val, fmscale scale, int use);
```

Unpacks packed data into the geophysical values. If e.g. AVHRR bi-directional reflectance or brightness temperatures are stored as unsigned short or integer values by the process generating the files, these values can be unpacked using this function. Typically brightness-temperatures are returned as Kelvin and bi-directional reflectance as percentage (0-100), but this is determined by the process encoding the data (e.g. KSPT MEOS, PolsatProc, etc.).

4.4.7 unpack_fmdatafield_ushort

```
float unpack_fmdatafield_ushort(unsigned short val, fmscale scale,
int use);
```

See 4.4.6 for details.

4.4.8 unpack_fmdatafield_short

```
float unpack_fmdatafield_short(short val, fmscale scale, int use);
```

See 4.4.6 for details.

4.4.9 unpack_fmdatafield_uint

```
float unpack_fmdatafield_uint(unsigned int val, fmscale scale, int
use);
```

See 4.4.6 for details.

4.4.10 unpack_fmdatafield_int

```
float unpack_fmdatafield_int(int val, fmscale scale, int use);
```

See 4.4.6 for details.

4.5 Color map specification for raster data

4.5.1 fmheatmap

```
void fmheatmap(int noclasses,
        unsigned short red[256],
        unsigned short green[256],
        unsigned short blue[256]);
```

Create a 24 bit colormap. This one looks like a heat topography, varying between red and blue. The number of classes selected will affect the difference between the various colours.

4.5.2 fmmapb2g

```
void fmmapb2g(int noclasses,
        unsigned short red[256],
        unsigned short green[256],
        unsigned short blue[256]);
```

Create a 24 bit colormap. This one goes from blue to green. The number of classes selected will affect the difference between the various colours.

4.5.3 fmmapr2g

```
void fmmapr2g(int noclasses,
        unsigned short red[256],
        unsigned short green[256],
        unsigned short blue[256]);
```

Create a 24 bit colormap. This one goes from red to green. The number of classes selected will affect the difference between the various colours.

4.6 Coordinate systems

4.6.1 fmgeo2ucs

```
fmucspos fmgeo2ucs(fmgeopos ll, fmprojspec myproj);
```

Given a geographical latitude and longitude in decimal degrees, and a code describing the map projection used, northings and eastings in kilometers relative to the North Pole are returned. myproj defines a map projection according to the definitions found in chapter 3.4 Coordinate systems.

4.6.2 fmucs2geo

```
fmgeopos fmucs2geo(fmucspos xy, fmprojspec myproj);
```

Given a User Coordinate System (UCS) coordinate (northings/eastings as defined above) and a map projection as defined above, a geographical latitude/longitude in decimal degrees is returned.

4.6.3 fmucsmeos2metno

```
fmucspos fmucsmeos2metno(fmucspos old);
```

Converts between UCS coordinates generated with the Kongeberg Spacetec standard definition (for Norway) and the standard for METNO.

4.6.4 fmucs2ind

```
fmindex fmucs2ind(fmucsref ref, fmucspos pos);
```

Converts between UCS coordinates and image index values. For image index values, Origo is in the upper left corner. All pixels are of the same size.

4.6.5 fmind2ucs

```
fmucspos fmind2ucs(fmucsref ref, fmindex ind);
```

As above, but reverse action.

4.6.6 fmind2geo

```
fmgeopos fmind2geo(fmucsref ref, fmprojspec myproj, fmindex ind);
```

Given a UCS reference description (from image metadata), a map projection description (see above) and image indices, a geographical latitude/longitude in decimal degrees is returned.

4.6.7 fmgeo2tile

```
int fmgeo2tile(fmgeopos ll);
```

This function is specifically created for the following tile definitions:

```
    fmucsref iref[4]={ /* these should probably be def somewhere else
*/
      {1.500,1.500,-335.000,-740.000,1200,1200}, /* nr */
      {1.500,1.500,-335.000,-2540.000,1200,1200}, /* ns */
      {1.500,1.500,-2135.000,-2540.000,1200,1200}, /* at */
      {1.500,1.500,-2135.000,-740.000,1200,1200} /* gr */
    };
```

Given a latitude/longitude in decimal degrees, the tile number is returned (returns -1 on failure). This is only done with the MI map projection (see above).

4.6.8 fmucs2diana

```
int fmucs2diana(fmucsref ucs, fmdianapos *nwp)
```

Computes NWP coordinate specifications from UCS. The UCS coordinate system is described above. The NWP coordinate system is a Cartesian coordinate system with Origo in the lower left corner of the image.

4.7 Observation geometry estimation from satellite subtrack

4.7.1 fmangest

```
fmbool fmangest(fmangreq rs, fmangles *ang);
```

Estimates the satellite and solar azimuth angles for a given position in latitude and longitude. Both the individual azimuth angles and the relative are estimated.

4.7.2 fmsubtrc

```
fmbool fmsubtrc(fmangreq rs, fmgeopos *p1, fmgeopos *p2);
```

Find the two closest satellite subtrack points to the target pixel, coordinates are returned as latitude, longitude, contrary to the old routine in libsatimg.

4.7.3 fmsubtrn

```
fmbool fmsubtrn(fmtriangpoints rs, fmgeopos *p);
```

To find the closest point (latitude, longitude) on a subsatellite track to the target point being examined. Spherical geometry is used. Input of the two closest given points on the satellite subtrack are input.

4.8 Byteswapping

4.8.1 fmshortbyteswap

```
void fmshortbyteswap(short int *data, int ndata);
```

To perform byte swapping when required between Little Endian (e.g. Intel platform) and Big Endian (e.g. SGI/IRIX). This function handles short integer swapping for vectors of length ndata..

4.8.2 fmintbyteswap

```
void fmintbyteswap(int *data, int ndata);
```

To perform byte swapping when required between Little Endian (e.g. Intel platform) and Big Endian (e.g. SGI/IRIX). This function handles integer swapping for vectors of length ndata..

4.8.3 fmbs_int

```
void fmbs_int(int *data);
```

To perform byte swapping when required between Little Endian (e.g. Intel platform) and Big Endian (e.g. SGI/IRIX). This function handles integer swapping for single values.

4.8.4 fmbs_short

```
void fmbs_short(short int *data);
```

To perform byte swapping when required between Little Endian (e.g. Intel platform) and Big Endian (e.g. SGI/IRIX). This function handles integer swapping for single values.

4.9 Radiance related functions

4.9.1 fmesd

```
double fmesd(int doy);
```

Estimates the variation of the distance between the Sun and the Earh throughout the year

using the equations specified in Paltridge and Platt (1976) at page 57. Day of Year (doy) should be in the range 0-364.

4.9.2 fmdeclination

```
double fmdeclination(int doy);
```

To estimate the declination. This varies between -23.5 degrees on December 22 and +23.5 on June 22. This function is base upon the equation given in Paltridge and Platt (1976), page 63. One should check the behaviour of gmtime when timezone is not set to UTC. The input doy variable is expected to be in the range 0 to 364 (January 1 and December 31 respectively).

4.9.3 fmequationoftime

```
fmsec1970 fmequationoftime(int doy);
```

To estimate the difference between local time and true solar time. Local time is the time transformed from some time zone dependency to the local position, equation of time compensates for the irregular behaviour of the Earth's orbit around the Sun. This function operates on time specified as seconds. The equation of time depends on the day of the year. The current implementation is given in at page 63 in Paltridge and Platt (1976). That equation is specified in radians, this is converted to seconds within this code. Day number of the year is expected to be generated by gmtime and to be in the range 0 to 364.

4.9.4 fmsolarzenith

```
double fmsolarzenith(fmsec1970 tst, fmgeopos gpos);
```

Estimates the solar zenith angle for a given position in latitude and longitude. It requires true solar time for the position specified as seconds since 1970 and latitude in decimal degrees as input.

4.9.5 fmbidirref

```
float fmbidirref(int jd, float soz, float alb);
```

To convert AVHRR albedo measurements to bi-directional reflectivities according to (more or less) standard texts by correcting it for the varying distance between the Earth and the Sun and transfer it from Sun at zenith to the actual Solar zenith angle. It is assumed that data from the MEOS system are converted from scaled integer to albedo values in the range 0-100 before entering this routine. The routine will return a value in the range 0-100 if input is in the range 0-100 or in the range 0-1 if the input is in the range 0-1. It is assumed that day of year is input in the range 1-365 and solar zenith angle in degrees.

See HTML paper by Rao and Chen concerning recalibration of NOAA-14 and CSIRO documentation concerning bi-directional reflectance.

4.9.6 fmtoairrad

```
double fmtoairrad(fmsec1970 tst, fmgeopos gpos, double s0);
```

To estimate the solar irradiance at Top of the Atmosphere (TOA). This function operates on time specified as seconds since 1970, i.e. standard UNIX time but specified as fmsec1970 and manipulated through the fmtime code.

4.10 String handling functions

4.10.1 fmremovenewline

```
void fmremovenewline(char *mystr);
```

Chops of newline (`'\n'`) characters within character strings and replaces them with `'\0'`.

4.10.2 fmstrsplit

```
void fmstrsplit(char *origString, const char *sep,
      unsigned int *nbTokens,char ***tokens);
```

Takes a string (e.g "`first:second:third`") and a separator ("`:`") and parses through the original string to count the number of tokens (3) and give access to each of them, as individual strings, in array `tokens` ("`first`","`second`" and "`third`").

4.10.3 fmstrjoin

```
char *fmstrjoin(size_t nbSubStrings, char *subStrings[], char sep);
```

Join several substrings (in array of strings subStrings) into a unique one, using the delimiter provided by the user. The resulting string is allocated inside this routine and should thus be released at a later stage.

4.10.4 fmbasename

```
void fmbasename(char *fullpathtofile, char **dirEnd, char
**fnameBeg);
```

Takes a string containing the full path name to a file (e.g. `/fou/sat/thomasl/test.txt`) and locate the two pointer parameters so that: `dirEnd` points to the last character of the path ('`l`') and `fnameBeg` points to the first character of the file name ('`t`'). Delimiter is '`/`' (slash). Multiple delimiter are correctly treated (e.g. `/fou/sat/thomasl//test.txt`). It should be noted that no memory is copied or allocated, pointers are only moved around.

4.10.5 fmgrep

```
fmbool fmgrep(char *needle, const char *haystack);
```

Implementation of a `grep` like function. Currently this is a prototype which supports up to 10 tokens. Use * to indicate whether the text searched for is appearing at the beginning or the end of the haystack.

4.10.6 fmgrepstart

```
fmbool fmgrepstart(char *needle, const char *haystack);
```

This function is mainly an internal function used by the utility function `fmgrep`.

4.10.7 fmgrepend

```
fmbool fmgrepend(char *needle, const char *haystack);
```

This function is mainly an internal function used by the utility function `fmgrep`.

4.10.8 fmstrlen

```
int fmstrlen(char *mystring);
```

Retrieve the length of the string examined.

4.11 Time handling functions

Some of the functions below (e.g. tofmsec1970 and ymdhms2fmsec1970) might behave odd when operated with time zones "Europe/London", "Europe/Dublin", "Europe/Lisbon", "Singapore", "GB", "GB-Eire", and "US/Alaska" on Red Hat and Fedora Core computers.. Thanks to John Stark at UKMO for pointing this out.

Whether this is a problem at other Linux distributions as well is not known.

4.11.1 check_fmtime

```
int check_fmtime(fmtime *fmt, int verbose);
```

Returns FM_OK if the fmtime object corresponds to a correct date, included that the number of days do not exceed the maximum allowed for the given month (28,29,30, or 31). Returns FM_IO_ERR if an error is found.

4.11.2 is_leap_year

```
int is_leap_year(int year);
```

Returns 1 if year is a leap year, 0 otherwise.

4.11.3 tofmtime

```
int tofmtime(fmsec1970 secs, fmtime *fmt);
```

Converts from fmsec1970 to `fmtime` structure.

4.11.4 tofmsec1970

```
fmsec1970 tofmsec1970(fmtime t);
```

Converts between fmtime and seconds since 1970 as represented by fmsec1970. The latter basically standard UNIX/LINUX time but handled by fmtime functions.

Returns **-1** if the time coded in the fmtime object is not valid.

4.11.5 fmstring2fmtime

```
int fmstring2fmtime(char date[], char fmt[], fmtime *res);
```

~~This routine is only available inside `fmtime.c` and is thus not intended for users but rather for developers of `libfmutil`.~~ This routine is now exported in fmutil.h.

Convert string `date` using format `fmt` into a `fmtime` object. For example:

```
if (FM_OK != fmstring2fmtime(“2008/02/16 12.10”,”YYYY/MM/DD hh.mm”,&time)) {
       // error handling goes here;
}
```

will achieve the correct conversion. Fields which are not specified in the date string are set to 0 (e.g. the seconds in the previous example). Codes are `YYYY`, `MM`, `DD`, `DOY`, `hh`, `mm`, `ss`. Note that the separators between the date/time fields are not checked for validity against those of the format string.

Returns FM_OK if the conversion went fine, and FM_IO_ERR otherwise.

This routine now checks for validity of the date string: month numbers greater than 12, or day numbers over 31 will raise an error. This is a new behaviour (Nov 2012) as previous versions of the routine were happy as long as the date string was syntactically correct.

4.11.6 ymdhms2fmsec1970

```
fmsec1970 ymdhms2fmsec1970(char *str,int mode);
```

Converts character strings of the type "20070502123000" to `fmsec1970` values. If `mode` differs from 0, the results are printed at standard out.

4.11.7 ymdhms2fmsec1970_alt

Same as `ymdhms2fmsec1970` but uses `fmstring2fmtime`.

4.11.8 ymdh2fmsec1970

```
fmsec1970 ymdh2fmsec1970(char *str,int mode);
```

Converts character strings of the type "2007050212" to `fmsec1970` values. If `mode` differs from 0, the results are printed at standard out.

4.11.9 ymdh2fmsec1970_alt

Same as `ymdh2fmsec1970` but uses `fmstring2fmtime`.

4.11.10 fmsec19702isodatetime

```
void fmsec19702isodatetime(fmsec1970 secs, char *str);
```

Convert `fmsec1970` value to 'iso' date and time format as used in the OSISAF. This format is `YYYY-MM-DDThh:mm:ssZ` with litteral 'T' and 'Z'. The output character string `str` has to be allocated beforehand. The character length (`'\0'` **not** included) is defined by `FMUTIL_ISODATETIME_LENGTH` in file `fmtime.h`.

4.11.11 isodatetime2fmsec1970

```
fmsec1970 isodatetime2fmsec1970(char *str);
```

Convert a `YYYY-MM-DDThh:mm:ssZ` (with litteral 'T' and 'Z') to `fmsec1970` value. Extensive checks are performed to ensure the validity of the input string. Routine returns -1 if the format is not valid or not describing a valid date.

4.11.12 isodatetime2fmsec1970_alt

Same as `isodatetime2fmsec1970` but uses `fmstring2fmtime`.

4.11.13 fmsec19702CFepoch

```
void fmsec19702CFepoch(fmsec1970 secs, char *str);
```

Convert `fmsec1970` value to 'CFepoch' date and time format as used in the OSISAF. This format is 'YYYY-MM-DD hh:mm:ss'. The output character string `str` has to be allocated beforehand. The character length (`'\0'` **not** included) is defined by `FMUTIL_CFEPOCH_LENGTH` in file `fmtime.h`.

4.11.14 CFepoch2fmsec1970

```
fmsec1970 CFeopch2fmsec1970(char *str);
```

Convert a 'YYYY-MM-DD hh:mm:ss' datestring into an `fmsec1970` value. Extensive checks are performed to ensure the validity of the input string. Routine returns -1 if the format is not valid or not describing a valid date. Makes use of `fmstring2fmtime`.

4.11.15 correct_time_from_unit

```
int correct_time_from_unit(double amount, char *str,fmsec1970 *secs);
```

Given an “amount” and a unit string (e.g. “seconds since YYYY-MM-DD hh:mm:ss”), computes and returns the number of seconds since 1 January 1970.

Beware that a prefix or postfix may be added to this function in the future to achieve a defined namespace!

4.11.16 fmutc2tst

```
fmsec1970 fmutc2tst(fmsec1970 fmutc, double lon);
```

To convert from UTC time to True Solar Time.

4.11.17 fmcet2tst

```
fmsec1970 fmcet2tst(fmsec1970 cet, double lon);
```

To convert from CET time to True Solar Time.

4.11.18 fmloncorr

```
fmsec1970 fmloncorr(double meridian, double lon);
```

Time deviation in true solar times due to longitudes when using time zones.

4.11.19 mjd19502fmtime

```
void mjd19502fmtime(double mjd, fmtime *cal);
```

Functions that convert from modified julian date referenced to 1. January 1950 to fmtime structures.

4.11.20 fmtime2mjd1950

```
void fmtime2mjd1950(fmtime cal, double *mjd);
```

Functions that convert to modified julian date referenced to 1. January 1950 from fmtime

structures.

4.11.21 fmhourangle

```
double fmhourangle(double tst);
```

To estimate the hour angle.

4.11.22 fmmeosdate2fmtime

```
int fmmeosdate2fmtime(char s[], fmtime *d);
```

To convert the date in MEOS MPH structure to midate format.

4.11.23 fmdayofyear

```
int fmdayofyear(fmtime md);
```

Computes Julian day number (day of the year in the range 1-365).

4.11.24 fmdoy2mmdd

```
int fmdoy2mmdd(int year, int doy, int *m2, int *d2);
```

Decode a DOY in a given year to a day of month (d2) and a month number (m2).

Returns FM_OK if conversion went fine, FM_IO_ERR otherwise.

4.12 Angular functions

4.12.1 fmrad2deg

```
double fmrad2deg(double radians);
```

Converts Radians to degrees.

4.12.2 fmdeg2rad

```
double fmdeg2rad(double degrees);
```

Converts degrees to Radians.

4.13 General algorithms

4.13.1 fmselalg

```
short fmselalg(fmtime d, fmucsref upos, float hmax, float hmin);
```

Based upon the UTC time of the image and the coordinates of the 4 corners, this algorithm determines whether daytime, nighttime or twilight algorithms should be used.