Bugs and Suggestions

Please notify us of any bugs you have found in our software and any suggestions you have for future releases or products.

Using the report form below, mail user feedback, bugs, or software suggestions via U.S. mail to:

NCSA Software Development - HDF
152 Computing Applications Bldg.
605 East Springfield Avenue
Champaign, IL 61820

Send reports regarding bugs via electronic mail to:

bugs@ncsa.uiuc.edu

Send reports regarding software suggestions or comments via electronic mail to:

softdev@ncsa.uiuc.edu

Name:

Institution:

Address (Electronic):

Address (U.S. Mail):

Telephone: (_____) _____ - _____.

Version of NCSA HDF: **Type machine:**

Version of system software:

Suggestion or description of problem:

Offset

A pointer to a specific location in the file relative to the first byte of the file. For purposes of numbering, the offset is the byte in the file *after* the offset (e.g., the first byte in the file is offset 0).

Raster Image Set (RIS)

An HDF set that provides a general framework for storing raster images and other data related to the display of a raster image.

Raster-8 Set

An HDF set that provides the basic information needed to store 8-bit raster images. The raster-8 set is supported, but has been superseded by the more general raster image set.

Reference Number

A number that distinguishes a data element from others with the same tag within the file. A reference number may *never* be zero.

Scientific Dataset (SDS)

An HDF set for storing scientific data organized in rectangular grid arrays, together with such information as array rank and dimensions, labels and units for interpreting dependent and independent variables and axis scales.

Set

A collection of related data objects that are grouped together to give full information about a particular type of application. The exact collection of objects that are stored together can vary, but for a given type of set certain object must always be included.

String

A series of ASCII bytes (fewer than 256) used as a text message. Strings always end with an ASCII NULL character and do not contain end-of-line characters. The purpose of the string type is to allow the unique identification of data elements, tags, and other quantities by a name.

Tag

A number used to designate what kind of data is contained in a data element. This specification defines a particular syntax and semantics for each tag number. Tags are also usually given a name programs can refer to instead of the number. A tag may never be zero.

Text

A series of ASCII bytes of any length used as a text message. Like strings, text always ends with a NULL character. Unlike strings, text may contain end-of-lines that are symbolized by the ASCII CR-LF pair. The purpose of the text type is to allow for detailed descriptions of data elements, tags, and other quantities.

Glossary

Calling Interface

A collection of related subroutines and perhaps utilities that provide users with needed functionality for working with a particular type of set. For each supported set, there is a corresponding calling interface.

Data Descriptor (DD)

A record containing the data identifier (tag/ref), the offset to the data, and the length of the data.

Data Descriptor Block (DD Block)

A series of data descriptors placed in the file and prefixed by a data descriptor header. All of the descriptors are assumed to contain significant data unless they have a tag which is equal to NULL (no data).

Data Descriptor Header (DDH)

A record that prefixes the data descriptor block. It has two fields: a size field giving the number of DDs in the following block, and a link field giving the offset of the next DD block, if there is one, or 0, if there is none.

Data Element

One unit of data in the file. Each data element has a unique data identifier and data descriptor entry which points to it.

Data Identifier (DI)

A quantity that uniquely defines a data element from all other data elements within a file. It is the concatenation of the reference number after the tag.

Data Object

Basic unit of information in HDF file, consisting of a data descriptor and the data element that it points to.

File Header (FH)

A four-byte magic number that indicates that the file is an HDF file. The contents of the file header is the ASCII code for the four characters ^N, ^C, ^S, and ^A.

Group

A collection of data identifiers (tag/refs) that identify all of the objects contained in a given instance of a set.

Length

The number of bytes occupied by a data element.

```
#define DFE_FNF        -1      /* File not found error */
#define DFE_DENIED             -2      /* Access to file denied */
#define DFE_ALROPEN            -3      /* File already open */
#define DFE_TOOMANY            -4      /* Too Many DF's or files open */
#define DFE_BADNAME            -5      /* Bad file name on open */
#define DFE_BADACC             -6      /* Bad file access mode */
#define DFE_BADOPEN            -7      /* Other open error */
#define DFE_NOTOPEN            -8      /* File can't be closed: it isn't open */
#define DFE_CANTCLOSE          -9      /* fclose wouldn't work! */
#define DFE_DFNULL             -10     /* DF is a null pointer */
#define DFE_ILLTYPE            -11     /* DF has an illegal type: internal error */
#define DFE_UNSUPPORTED        -12     /* Feature not currently supported */
#define DFE_BADDDLIST          -13     /* No DD list: internal error */
#define DFE_NOTDFFILE          -14     /* Not a DF file and it is not 0 length */
#define DFE_SEEDTWICE          -15     /* DD list already seeded: internal error */
#define DFE_NOSPACE            -16     /* Malloc failed */
#define DFE_NOSUCHTAG          -17     /* No such tag in the file: search failed*/
#define DFE_READERROR          -18     /* There was a read error */
#define DFE_WRITEERROR -19     /* There was a write error */
#define DFE_SEEKERROR          -20     /* There was a seek error */
#define DFE_NOFREEDD           -21     /* No free DD's left: internal error */
#define DFE_BADTAG             -22     /* illegal WILDCARD tag */
#define DFE_BADREF             -23     /* illegal WILDCARD reference # */
#define DFE_RDONLY             -24     /* The DF is read only */
#define DFE_BADCALL            -25     /* Calls in wrong order */
#define DFE_BADPTR             -26     /* NULL ptr argument */
#define DFE_BADLEN             -27     /* negative len specified */
#define DFE_BADSEEK            -28     /* Attempt to seek past end of element */
#define DFE_NOMATCH            -29     /* No (more) DDs which match specified tag/ref */
#define DFE_NOTINSET           -30     /* Warning: Set contained unknown tag: ignored */
#define DFE_BADDIM             -31     /* negative or zero dimensions specified */
#define DFE_BADOFFSET          -32     /* Illegal offset specified */
#define DFE_BADSCHEME          -33     /* Unknown compression scheme specified */
#define DFE_NODIM              -34     /* No dimension record associated with image */
#define DFE_NOTENOUGH          -35     /* space provided insufficient for size of data
```

Figure B.2 C Header File: df.h (Continued)

```
*/
#define DFE_NOVALS             -36     /* Values not available */
#define DFE_CORRUPT            -37     /* File is corrupted */
#define DFE_BADCONV            -37     /* Don't know how to convert data type */
#define DFE_BADFP              -38     /* The file contained an illegal floating-point no*/

/*----------------------------------------*/
/*             Logical Constants                                          */

#define DFACC_READ             1       /* Read Access */
#define DFACC_WRITE            2       /* Write Access */
#define DFACC_CREATE           4       /* force file to be created */
#define DFACC_ALL              7       /* logical and of all the above values */

#endif DFTAG_NULL
```

```
        /* utility set */
#define DFTAG_FID      100          /* File identifier */
#define DFTAG_FD       101          /* File description */
#define DFTAG_TID      102          /* Tag identifier */
#define DFTAG_TD       103          /* Tag descriptor */
#define DFTAG_DIL      104          /* data identifier label */
#define DFTAG_DIA      105          /* data identifier annotation */
#define DFTAG_NT       106          /* number type */
#define DFTAG_MT       107          /* machine type */

        /* raster-8 set */
#define DFTAG_ID8      200          /* 8-bit Image dimension */
#define DFTAG_IP8      201          /* 8-bit Image palette */
#define DFTAG_RI8      202          /* Raster-8 image */
#define DFTAG_CI8      203          /* RLE compressed 8-bit image */
#define DFTAG_II8      204          /* IMCOMP compressed 8-bit image */

        /* Raster Image set */
#define DFTAG_ID       300          /* Image DimRec */
#define DFTAG_LUT      301          /* Image Palette */
#define DFTAG_RI       302          /* Raster Image */
#define DFTAG_CI       303          /* Compressed Image */

#define DFTAG_RIG      306          /* Raster Image Group */
#define DFTAG_LD       307          /* Palette DimRec */
#define DFTAG_MD       308          /* Matte DimRec */
#define DFTAG_MA       309          /* Matte Data */
#define DFTAG_CCN      310          /* color correction */
#define DFTAG_CFM      311          /* color format */
#define DFTAG_AR       312          /* aspect ratio */

#define DFTAG_DRAW     400          /* Draw these images in sequence */
#define DFTAG_RUN      401          /* run this as a program/script */
```

Figure B.2 C Header File: df.h (Continued)

```
#define DFTAG_XYP      500          /* x-y position */
#define DFTAG_MTO      501          /* machine-type override */

        /* Tektronix */
#define DFTAG_T14      602          /* TEK 4014 data */
#define DFTAG_T105     603          /* TEK 4105 data */

        /* Scientific Dataset */
#define DFTAG_SDG      700          /* Scientific Data Group */
#define DFTAG_SDD      701          /* Scientific Data DimRec */
#define DFTAG_SD       702          /* Scientific Data */
#define DFTAG_SDS      703          /* Scales */
#define DFTAG_SDL      704          /* Labels */
#define DFTAG_SDU      705          /* Units */
#define DFTAG_SDF      706          /* Formats */
#define DFTAG_SDM      707          /* Max/Min */
#define DFTAG_SDC      708          /* Coord sys */

        /* compression schemes */
#define DFTAG_RLE      11           /* run length encoding */
#define DFTAG_IMCOMP 12             /* IMCOMP compression */

/*--------------------------------------*/
/*            Error Return Codes                                    */

#define DFE_NOERROR          0      /* No error */
```

```
        int
                type,           /* 0= not in use, 1= normal, -1 = multiple */
                                /* this is a hook for when multiple files are open */
                access,/* permitted access types: 0=none, 1=r, 2=w, 3=r/w */
                changed,        /* True if anything in DDs modified since last write */
                last_tag,       /* Last tag searched for by DFfind */
                last_ref,       /* Last reference number searched for */
                last_dd,        /* see last_dle */
                defdds,/* default number of DD's in each block */
                up_access;      /* access permissions to element being read/updated */
                                /* Used by DFstart */
        DFdd
                *up_dd;/* DD of element being read/updated, used by DFstart */
                                /* file handle is a file pointer or file descriptor */
                                /* depending whether we use buffered or unbuffered i/o */

#ifdef DF_BUFFIO
        FILE *                  /* file pointer */
#else DF_BUFFIO
        int                     /* file descriptor */
#endif DF_BUFFIO
                file;           /* File handle for real file */
} DF;


typedef struct DFdi { /* data identifier: specifies data element uniquely */
        uint16 tag;
        uint16 ref;
} DFdi;



typedef struct DFdata {         /* structure for returning status information */
        int version;            /* version number of program */
} DFdata;
```

Figure B.2 C Header File: df.h (Continued)

```
/*————————————————————————————————————*/
/*              Procedure types                                         */

DF *DFopen();
int32 DFgetelement();
int32 DFread();
int32 DFseek();
int32 DFwrite();

/*————————————————————————————————————*/
/*              Global Variables                                        */

#ifndef DFMASTER
extern
#endif DFMASTER
int
        DFerror;                        /* Error code for DF routines */

/*————————————————————————————————————*/
/*              Tag Definitions                                             */

#define DFREF_WILDCARD 0        /* wildcard ref for searches */

#define DFTAG_WILDCARD 0        /* wildcard tag for searches */
#define DFTAG_NULL     1                /* empty DD */
```

```
 * Invokes:    dfi.h
 * Contents:
 *      Structure definitions: DFddh, DFdd, DFdesc, DFdle, DF, DFdi, DFdata
 *      Procedure type definitions
 *      Global variables
 *      Tag definitions
 *  Error return codes
 *      Logical constants
 * Remarks: This file is included with user programs
 *             Since it includes stdio.h etc., do not include these after df.h
 *————————————————————————————————————*/


#ifndef DFTAG_NULL                          /* avoid re-inclusion */

/* include DF (internal) header information */
#include "dfi.h"

/*————————————————————————————————————*/
/*Type declarations                                                    */

typedef struct DFddh {              /* format of data descriptor headers as in file */
       int16
              dds;                  /* number of dds in header block */
       int32
              next;                 /* offset of next header block */
       } DFddh;


typedef struct
       uint16
              tag,                  /* data tag */
              ref;                  /* data reference number */
       int32
              offset,        /* offset of data element in file */
              length;        /* number of bytes */
       } DFdd;
```

Figure B.2 C Header File: df.h (Continued)

```
       /* descriptor structure is same as dd structure.  ###Note: may be changed */
#define DFdesc DFdd


       /* DLE is the internal structure that stores data descriptor information */
       /* It is a linked list of DDs */
typedef struct DFdle {              /* Data List element */
       struct DFdle
              *next;                /* link to next dle */
       DFddh
              ddh;                  /* To store headers */
       DFdd
              dd[1];                /* dummy size */
       } DFdle;


       /* DF is the internal structure associated with each DF file */
       /* It holds information associated with the file as a whole */
       /* ### Note: there are hooks for having multiple DF files open at a time */
typedef struct DF {
       DFdle
              *list,         /* Pointer to the DLE list */
              *last_dle;     /* last_dle and last_dd are used in searches to indicate
                                    element returned by previous call to DFfind */
```

```
#define DFE_NOTOPEN    -8
#define DFE_CANTCLOSE  -9
#define DFE_DFNULL     -10
#define DFE_ILLTYPE    -11
#define DFE_UNSUPPORTED       -12
#define DFE_BADDDLIST         -13
#define DFE_NOTDFFILE         -14
#define DFE_SEEDTWICE         -15
#define DFE_NOSPACE           -16
#define DFE_READERROR         -18
#define DFE_WRITEERROR -19
#define DFE_SEEKERROR         -20
#define DFE_NOFREEDD          -21
#define DFE_BADTAG            -22
#define DFE_BADREF            -23
#define DFE_RDONLY            -24
#define DFE_BADCALL           -25
#define DFE_BADPTR            -26
#define DFE_BADLEN            -27
#define DFE_BADSEEK           -28
#define DFE_NOMATCH           -29
#define DFE_NOTINSET          -30
#define DFE_BADDIM            -31
#define DFE_BADOFFSET         -32
#define DFE_BADSCHEME         -33
#define DFE_NODIM             -34
#define DFE_NOTENOUGH         -35

C                                       Logical Constants

#define DFACC_READ            1
#define DFACC_WRITE           2
#define DFACC_CREATE          4
#define DFACC_ALL             7

#endif DFTAG_NULL
```

Figure B.2 C Header File: df.h

```
/*****************************************************************************
*
*                                                    NCSA HDF version 2.0
*                                                         December 20, 1988
*
* NCSA HDF Version 2.0 source code and documentation are in the public domain.
* Specifically, we give to the public domain all rights for future licensing
* of the source code, all resale rights, and all publishing rights.
*
* We ask, but do not require, that the following message be included in all
* derived works:
*
* Portions developed at the National Center for Supercomputing Applications at
* the University of Illinois at Urbana-Champaign.
*
*
* THE UNIVERSITY OF ILLINOIS GIVES NO WARRANTY, EXPRESSED OR IMPLIED, FOR THE
* SOFTWARE AND/OR DOCUMENTATION PROVIDED, INCLUDING, WITHOUT LIMITATION,
* WARRANTY OF MERCHANTABILITY AND WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE
*
*****************************************************************************/

/*----------------------------------------
 * File:       df.h
 * Purpose:    header file for HDF routines
```

```
C * Error return codes
C *    Logical constants
C * Remarks: This file can be included with user programs
C *————————————————————————————*

#ifndef DFTAG_NULL

#define DFREF_WILDCARD 0

#define DFTAG_WILDCARD 0
#define DFTAG_NULL     1

#define DFTAG_FID      100
#define DFTAG_FD       101
#define DFTAG_TID      102
#define DFTAG_TD       103
#define DFTAG_DIL      104
#define DFTAG_DIA      105
#define DFTAG_NT       106
#define DFTAG_MT       107

#define DFTAG_ID8      200
#define DFTAG_IP8      201
#define DFTAG_RI8      202
#define DFTAG_CI8      203
#define DFTAG_II8      204

#define DFTAG_ID       300
#define DFTAG_LUT      301
#define DFTAG_RI       302
#define DFTAG_CI       303

#define DFTAG_RIG      306
#define DFTAG_LD       307
#define DFTAG_MD       308
#define DFTAG_MA       309
```

Figure B.1 Fortran Header File: dfF.h (Continued)

```
#define DFTAG_CCN      310
#define DFTAG_CFM      311
#define DFTAG_AR       312

#define DFTAG_DRAW     400
#define DFTAG_RUN      401

#define DFTAG_XYP      500
#define DFTAG_MTO      501

#define DFTAG_T14      602
#define DFTAG_T105     603

#define DFTAG_RLE      11
#define DFTAG_IMCOMP 12

C                                   Error Return Codes

#define DFE_NOERROR    0
#define DFE_FNF -1
#define DFE_DENIED     -2
#define DFE_ALROPEN    -3
#define DFE_TOOMANY    -4
#define DFE_BADNAME    -5
#define DFE_BADACC     -6
```

Appendix B Header Files

Overview

This appendix includes the general header files used in compiling all HDF libraries. These files do *not* have to be included with most applications, but they could be usefully added to code that refers to specific tags by name, or code that is designed to be responsive to different error conditions. You get these files when you get the HDF source code.

You will see that the C header (df.h) shown in Figure B.2 is more completely commented than the Fortran header (dfF.h) shown in Figure B.1. So you may want to look at the C header for explanations of some definitions, even if you plan to use the #define statements from the Fortran header.

Other headers used for specific applications, such as the RIS interface and the SDS interface can be found by obtaining the source code from NCSA. They also do not normally have to be included with your source code.

Figure B.1 Fortran Header File: dfF.h

```
/*****************************************************************************
*
*                                                    NCSA HDF version 2.0
*                                                         December 20, 1988
*
* NCSA HDF Version 2.0 source code and documentation are in the public domain.
* Specifically, we give to the public domain all rights for future licensing
* of the source code, all resale rights, and all publishing rights.
*
* We ask, but do not require, that the following message be included in all
* derived works:
*
* Portions developed at the National Center for Supercomputing Applications at
* the University of Illinois at Urbana-Champaign.
*
* THE UNIVERSITY OF ILLINOIS GIVES NO WARRANTY, EXPRESSED OR IMPLIED, FOR THE
* SOFTWARE AND/OR DOCUMENTATION PROVIDED, INCLUDING, WITHOUT LIMITATION,
* WARRANTY OF MERCHANTABILITY AND WARRANTY OF FITNESS FOR A PARTICULAR PURPOSE
*
*****************************************************************************/

C *-----------------------------------------
C * File:      dfF.h
C * Purpose:   Fortran header file for HDF routines
C * Contents:
C *     Tag definitions
```

(708)

This tag points to a string specifying the coordinate system for the dataset. The string is terminated by a null byte.

SDT

Scientific data transpose
0 bytes
(709)

The presence of this tag indicates that the data pointed to by the SD tag is in column-major order, instead of the default row-major order. No data is associated with this tag.

This tag points to an array of scientific data. The type of the data may be specified by an NT tag included with the SDG. If there is no NT tag, the type of the data is floating-point in standard IEEE 32-bit format. The rank and dimensions must be stored as specified in the corresponding SDD tag.

SDS

Scientific data scales
n+4*(x+1+y+1+z+1+...) bytes (where n is the rank and x, y, z, etc. are the dimensions of the array.)
(703)

This tag points to the scales for the dataset. The first n bytes indicate whether there is a scale for the corresponding dimension (1=yes, 0=no). This is followed by the scale values for each dimension.

SDL

Scientific data labels
? bytes
(704)

This tag points to a list of labels for the data and each dimension of the dataset. Each label is a string terminated by a null byte.

SDU

Scientific data units
? bytes
(705)

This tag points to a list of strings specifying the units for the data and each dimension of the dataset. Each unit's string is terminated by a null byte.

SDF

Scientific data format
? bytes
(706)

This tag points to a list of strings specifying an output format for the data and each dimension of the dataset. Each format string is terminated by a null byte.

SDM

Scientific data max/min
8 bytes
(707)

This record contains the maximum and minimum data values in the dataset. It consists of two floating-point numbers.

SDC

Scientific data coordinates
? bytes

tion programs that deal with SDGs should read all of the elements of a SDG and process those identifiers which it can use. Even if an application cannot process *all* of the tags, the tags that it can use will be displayable.

Tag types that may appear in a SDG are listed in Table A.8.

Figure A.8 Possible Tag Types in an SDG

Tag	Description
SDD	scientific data dimension record (rank and dimensions)
SD	scientific data
SDS	scales
SDL	labels
SDU	units
SDF	formats
SDM	maximum and minimum values
SDC	coordinate system
SDT	transposition

Example

SDD, SD, SDM
A dimension record, the scientific data, and the maximum and minimum values of the data go together. The application reads the rank and dimensions from the dimension record, then reads the data array with those dimensions. If it needs maximum and minimum, it also reads them.

SDD

Scientific data dimension record
16 + 4**rank* bytes
(701)

This record defines the rank and dimensions of the array in the scientific dataset. The fields are defined as shown in Table A.9.

Table A.9 Scientific Data Dimension Record Fields

Field	Definition
Rank	16-bit integer
Dimension sizes	32-bit integers
Data NT (number type)	4 bytes
Scale NTs	*rank*4 bytes

For example, an SDD for a 500x600x3 array of floating-point numbers would have the following values and components.

Rank: 3
Dimensions: 500, 600, and 3.
One data NT
Three scale NTs

SD

Scientific data
4**x***y***z**... bytes (where *x*, *y*, *z*, etc. are the dimensions)
(702)

Composite Raster Image Tag

XYP

XY position
8 bytes
(500)

An XY position is used in composites and other groups to indicate an XY position on the screen. For this, (0,0) is the lower left, X is the number of pixels to the right along the horizontal axis and Y is the number of pixels on the vertical axis. The X and Y pixel dimensions are given as two 32-bit integers.

For example, if XYP is present inside an RIG, the XYP refers to the position of the lower left corner of the raster image on the screen.

See also: DRAW (Drawing from a list of elements)

Vector Image Tags

T14

Tektronix 4014
? bytes
(602)

This tag points to a Tektronix 4014 data stream. The bytes in the data field, when read and sent to a Tektronix 4014 terminal, will display a vector image. Only the low seven bits of each byte are significant. There are no record markings or non-Tektronix codes in the data.

T105

Tektronix 4105
? bytes
(603)

This tag points to a Tektronix 4105 data stream. The bytes in the data field, when read and sent to a Tektronix 4105 terminal, will be displayed as a vector image. Only the low seven bits of each byte are significant. Some terminal emulators will not correctly interpret every feature of the Tektronix 4105 terminal, so you may wish to use only a subset of the possible Tektronix 4105 vector commands.

Scientific Dataset Tags

SDG

Scientific data group
*n**4 bytes (where *n* is the number of data objects in the group.)
(700)

The scientific data group (SDG) data is a list of data identifiers (tag/ref) that describe a scientific dataset. All of the members of the group provide information for correctly interpreting and displaying the data. Applica-

CFM

Color format
string
(311)

The color format is a clue to how each element of each pixel in a raster image. It is defined to be a string which is in all caps, and is one of the values shown in Table A.7.

Table A.7 Color Format String Values

String	Description
VALUE	psuedo-color, or just a value associated with the pixel
RGB	red, green, blue model
XYZ	color-space model
HSV	hue, saturation, value model
HSI	hue, saturation, intensity
SPECTRAL	spectral sampling method

AR

Aspect ratio
4 bytes
(312)

The data for this tag is the visual aspect ratio for this image. The image should be visually correct if displayed on a screen with this aspect ratio. The data consists of one floating-point number which represents width divided by height. An aspect ratio of 1.0 indicates a display with perfectly square pixels; 1.33 is a standard aspect ratio used by many monitors.

Composite Image Tag

DRAW

Draw
*n**4 bytes (where *n* is the number of data objects that comprise the composite image.)
(400)

The data for this tag is a list of data identifiers (tag/ref) which define a composite image. Each member of the DRAW data should be displayed, in order, on the screen. This can be used to indicate several RIGs which should be displayed simultaneously, or even include vector overlays, like T14, which should be placed on top of a RIG.

Some of the elements in a DRAW list will be instructions about how images are to be composited (XOR, source put, anti-aliasing, etc.). These are defined as individual tags.

See also:

RIG (Raster image display)
T14 (Tektronix 4014 for overlay)

In this case, NT specifies an 8-bit integer. There are three elements per node—one each for red, green, and blue.

Interlace=0 indicates that the RGB values are not separated.
Compression=0 means no compression scheme is used.

RI

Raster image
*r*c* bytes (where *r* and *c* are the number of rows and columns in the image, respectively.)
(302)

This tag points to raster image data. It must be stored as specified in an ID tag.

Interlace=0 means each pixel is contiguous.
Interlace=1 means each element is grouped by scan lines.
Interlace=2 means each element is grouped by planes.

LUT

Lookup table
? bytes
(301)

The LUT, sometimes called a palette, is used by many kinds of hardware to assign RGB colors or HSV colors to data values. When a raster image consists of data values which are going to be interpreted through hardware with a LUT capability, the LUT should be loaded along with the image.

The most common lookup table will have X dimension=256 and Y dimension=1 with three elements per entry, one each for red, green, and blue. The interlace will be either 0, where the LUT values are given RGB, RGB, RGB..., or 1, where the LUT values are given as 256 reds, 256 greens, 256 blues.

CCN

Color correction
52 bytes (usually)
(310)

Color correction specifies the Gamma correction for the image and color primaries for the generation of the image. The fields, in order, are shown in Table A.6.

Table A.6 Color Correction Field Types

Correction	Field Type
Gamma	float
Red (X,Y,Z)	three floats
Green (X,Y,Z)	three floats
Blue (X,Y,Z)	three floats
White (X,Y,Z)	three floats

Table A.4 Possible Tag Types in an RIG

Tag	Description
ID	image dimension
RI	raster image
XYP	X-Y position
LD	LUT dimension
LUT	color lookup table
MD	matte channel dimension
MA	matte channel
CCN	color correction
CFM	color format
AR	aspect ratio
MTO	machine-type override

Example

ID, RI, LD, LUT

An image dimension record, the raster image, an LUT dimension and the LUT go together. The application reads the image dimensions, then reads the image with those dimensions. It also reads the lookup table according to its dimensions and displays the corresponding image.

ID, LD, MD

Image dimension
20 bytes
(300)

LUT dimension
20 bytes
(307)

Matte dimension
20 bytes
(308)

These three dimension records have exactly the same format. They define the dimensions of the 2D array to which each refers. ID specifies the dimensions of an RI tag, LD specifies the dimensions of an LUT tag, and MD specifies the dimensions of a MA tag. The fields are defined as shown in Table A.5.

Table A.5 Dimension Record Fields

Field	Definition
X-dimension (width)	32-bit integer
Y-dimension (height)	32-bit integer
NT/ref (element type)	tag and ref=32 bits
Elements per node	16-bit integer
Interlace scheme	16-bit integer (0, 1, or 2)
Compression tag	tag and ref=32 bits

For example, a 512x256 row-wise 24-bit raster image with each pixel stored as RGB bytes would have the following values:

X:512, Y:256

(201)

The data for this tag consists of 256 triples of three bytes. The bytes are for the red, green, and blue elements of the 256-byte palette respectively. The first triple is palette entry 0 and the last is palette entry 255.

RI8

Raster image-8
*r***c* bytes (where *r* and *c* are the number of rows and columns in the image, respectively.)
(202)

The data for this tag is a row-wise representation of the elementary 8-bit image data. The data is stored width-first (hence row-wise) and is 8 bits per pixel. The first byte of data represents the pixel in the upper-left hand corner of the image.

CI8

Compressed image-8
? bytes
(203)

The data for this tag is a row-wise representation of the elementary 8-bit image data. Each row is compressed using the following run-length encoding where *n* is the low seven bits of the byte. The high bit represents whether the following *n* character will be reproduced exactly (high bit=0) or whether the following character will be reproduced *n* times (high bit=1). Since CI8 and RI8 are basically interchangeable, it is suggested that you not have a CI8 and a RI8 that have the same reference number.

II8

IMCOMP image-8
? bytes
(204)

The data for this tag is a 4:1 compressed 8-bit image, using the IMCOMP compression scheme.

General Raster Image Tags

RIG

Raster image group
*n**4 bytes (where *n* is the number of data objects in the group.)
(306)

The raster image group (RIG) data is a list of data identifiers (tag/ref) that describe a raster image. All of the members of the group are required to display the image correctly. Application programs that deal with RIGs should read all the elements of a RIG and process those identifiers which it can display correctly. Even if the application cannot process *all* of the tags, the tags that it can process will be displayable.

Tag types that may appear in a RIG are listed in Table A.4.

recommend that all HDF files contain an MT tag for maximum portability.

Possible machine types are shown in Table A.3.

Table A.3 Possible Machine Types

Type	Possible Machines
floats	IEEE32, VAX32, CRAY64
ints	VAX32, Intel16, Intel32, Motorola32, CRAY64
chars	ASCII, EBCDIC
double	IEEE64, VAX64, CRAY128

Each type is extensible as we find a need for new types.

RLE

Run length encoded data
0 bytes
(11)

This tag is used in the ID compression field and other places to indicate that an image or section of data is encoded with a run-length encoding scheme. The RLE method used is byte-wise. The low seven bits of the count byte indicate the number of bytes (n). The high bit of the count byte indicates whether the next byte should be replicated *n* times (high bit=1), or whether the next *n* bytes should be included as is (high bit=0).

See also: RIG (Raster Image Group)

IMC

IMCOMP compressed data
0 bytes
(12)

This tag is used in the ID compression field and other places to indicate that an image or section of data is encoded with an IMCOMP encoding scheme. This scheme is a 4:1 aerial averaging method which is easy to decompress. It counts color frequencies in 4x4 squares to optimize color sampling.

See also: RIG (Raster Image Group)

Raster-8 (8-Bit Only) Tags

ID8

Image dimension-8
4 bytes
(200)

The data for this tag consists of two 16-bit integers representing the width and height of an 8-bit raster image in bytes.

IP8

Image palette-8
768 bytes

Some possible values that may be included for each of the three types in the field CLASS are listed in Table A.2.

Table A.2 Number Type Values

Type	Possible Values
floats	IEEE floating-point, VAX floating-point, Cray floating-point
ints	VAX byte order, Intel byte order, Motorola byte order
chars	ASCII, EBCDIC

The number type flag is used by any other element in the file to indicate specifically what a numeric value looks like. Other tag types should contain a reference number pointer to an NT tag instead of containing their own number type definitions.

If an MT element is present in the file, then all NTs can be assumed to be of the appropriate default types for that machine, unless required by definition.

The definition of NT has brought up a unique situation where the definition of a generic, always applicable, set of fields is considered too difficult for implementation in the early version. Therefore, NCSA defined version 1 of the NT tag to contain only four fields. A draft of the version 2 tag definition starts with the same four fields and adds fields that can define types of numbers not currently listed.

Version 1 NT implementations should check the version number field only to confirm it, and must always write a 1 into this field. Version 2 implementations will have to be backward compatible in as many cases as possible.

MT

Machine type
0 bytes
(107)

The MT tag specifies that all unconstrained or partially constrained values in this HDF file are of the default type for that hardware. When the MT tag is set to VAX, for example, all integers will be assumed to be in VAX byte order unless specifically defined otherwise with an NT tag. Note that all of the headers and many tags, the whole raster-8 set for example, are defined with bit-wise precision and will not be overidden by the MT setting.

For MT, the reference field itself is the encoding of the MT information. The reference field is 16 bits, taken as four groups of four bits, specifying the types for unsigned char, unsigned int, float, and double respectively. This allows 16 generic specifications for each type.

To the user, these will be defined constants in df.h, specifying the proper descriptive numbers for Sun, VAX, Cray, Alliant, and other computer systems. If there is no MT tag in a file, the application may assume that the data in the file has been written on the local machine—assuming any portability problems are taken care of by the user. For this reason, we

The data for this tag is a text block which describes in relative detail the functionality and format of the tag which is indicated in the reference number. This tag is mainly intended to be used with user-defined tags and provides a medium for users to exchange files that include human-readable descriptions of the data.

It is important to provide everything that a programmer might need to know to read the data from your user-defined tag. At the minimum, you should specify everything you would need to know in order to retrieve your data at a later date if the original program were lost.

DIL

Data identifier label
string
(104)

The data for this tag is a data identifier, made up of a tag and reference number, followed by a string that the user wants to place in the file. The purpose of this tag is to associate the string with the data identifier as a label for whatever that data identifier points to in turn.

With DIL, any data identifier can be labeled. Each data identifier points to some data in the file. By including DILs, you can give any piece of data a label for future reference. For example, DIL is often used to give titles to images.

DIA

Data identifier annotation
text
(105)

The data for this tag is a data identifier, which is made up of a tag and a reference number, followed by a text block that the user wants to place in the file. Its purpose is to associate the text block with the data identifier as annotation for whatever that data identifier points to in turn.

With DIA, any data identifier can have a lengthy, user-written description of why that data is in the file. This will be used to include user comments about images, datasets, source code, and so forth.

NT

Number type
4 bytes
(106)

NT consists of four fields of 1 byte each, as shown in Table A.1.

Table A.1 Number Type Fields

Field	Contents
VERSION	version number of NT information, currently=1
TYPE	unsigned int, signed int, unsigned char, char, float, double
WIDTH	number of bits (assumed all significant)
CLASS	a generic value, with different interpretations depending on type: floating-point, integer, or character

Utility Tags

NULL

No data
0 bytes
(1)

This tag is used for place holding and to fill empty portions of the data descriptor block. The length and offset fields of a NULL DD must be equal to zero.

FID

File identifier
string
(100)

This tag points to a string which the user wants to associate with this file. The string is intended to be a user-supplied title for the file.

FD

File descriptor
text
(101)

This tag points to a block of text describing the overall file contents. The text can be any length, in the standard text format for HDF. Text is intended to be user-supplied comments about the file.

TID

Tag identifier
string
(102)

The data for this tag is a string that identifies the functionality of the tag indicated in the reference number. For example, the tag identifer for tag identifier would point to data that reads “tag identifier.” The reference number for the tag identifier is another tag number.

Many tags are identified in the HDF specification, so it is usually unnecessary to include their identifiers in the HDF file. But with user-defined tags or special-purpose tags, the only way for a human reader to diagnose what kind of data is stored in a file is to read tag identifiers. Use tag *descriptors* to define even more detail about your user-defined tags.

Note that with this tag you may make use of the user-defined tags to check for consistency. Although two persons may use the same user-defined tag, they probably will not use the same tag identifier.

TD

Tag descriptor
text
(103)

Appendix A NCSA HDF Tags

Overview

This appendix contains a complete list of HDF tags that have been assigned at NCSA as of March 1989. Each entry is to be interpreted as follows:

- The word in capital letters on the left is an acronym for the tag. In the NCSA software, the identifiers that refer to these names are preceded by DFTAG_ and are defined in the files df.h and dfF.h, which are listed in Appendix B.

- The three short lines at the beginning of each description uniquely identify the tag:

  The first line is the full name of the tag.

  The second line describes the type and (where possible) the amount of data in the corresponding data element. When the data element is a variable-sized data structure—such as text, a string, or a variable-sized array—the amount of data cannot be specified exactly. Where possible, a formula is given for estimating the amount of data. If the second line is "? bytes" it means that neither the size nor the structure of the data element can be specified.

  The third line gives the tag number.

- Following the three short lines is a full specification of the tag, including a description of the data element and a discussion of its intended use.

These listings are grouped approximately according to the roles that the tags play under the headings Utility Tags, Raster-8 Tags, General Raster Image Tags, and so forth. These groupings imply a general context for the use of each tag, but are not meant to restrict the use of the tags to any particular context.