Purpose: To provide status information about an HDF file.

Returns: 0 on success; -1 on failure.

This routine currently does nothing but return the version number of the HDF library. It is intended for later expansion.

Returns: 0 if it is an HDF file; -1 if not.

DFnewref

```
FORTRAN:
INTEGER FUNCTION DFnewref(dfile)

INTEGER dfile             - pointer to open HDF file
```

```
C:
uint16 DFnewref(dfile)

DF       *dfile       /* pointer to open HDF file */
```

Purpose: To get an unused reference number.

Returns: Unused ref number if successful; 0 on failure.

This routine is useful when you want to add a data element to an HDF file and you need a unique reference number. The value returned is a ref which is not used with any tag, except possibly `DFTAG_MT`.

DFnumber

```
FORTRAN:
INTEGER FUNCTION DFnewref(dfile)

INTEGER dfile             - pointer to open HDF file
```

```
C:
int DFnumber(dfile, tag)

DF       *dfile       /* pointer to open HDF file */
unit16   tag;         /* tag to count occurrences of */
```

Purpose: To return the number of occurrences of the given tag in the HDF file.

Returns: The number of occurrences of `tag` in the file if successful; -1 on error with DFerror set . If the `tag` is `DFTAG_WILDCARD`, `DFnumber` returns the occurrences of all tags.

DFstat

```
FORTRAN:
INTEGER FUNCTION DFstat(dfile, dfinfo)

INTEGER dfile             - pointer to an open HDF file
INTEGER dfinfo            - pointer to information on file
```

```
C:
int DFstat()
DF       *dfile       /* pointer to an open HDF file  */
struct DFdfinfo       /* pointer to information about file */
```

DFdel

FORTRAN:

```
INTEGER FUNCTION DFdel(dfile, tag, ref)

INTEGER      dfile      -   identifier of file access path
INTEGER      tag, ref   -   tag and ref of data to be deleted
```

C:

```
int DFdel(dfile, tag, ref)

DF       *dfile;      /* file access path */
uint16   tag, ref;    /* tag and ref of data to be deleted */
```

Purpose: To delete a tag/ref from the list of DDs.

Returns: 0 on success; -1 on failure.

The data to which this DD points is not affected by `DFdel`, but is reclaimed when the file is compacted.

If there are multiple references to this data, then the other references remain intact, and compaction does not affect the data itself. If there are no other references to a data element, `DFdel` may implicitly leave the data without any reference, making the data inaccessible.

Miscellaneous

DFerrno

FORTRAN:

```
INTEGER FUNCTION DFerrno()
```

C:

```
int DFerrno()
```

Purpose: To report the value of `DFerror`.

Returns: The value of `DFerror`.

DFishdf

FORTRAN:

```
INTEGER FUNCTION DFishdf(filename)

integer filename        - name of HDF file
```

C:

```
int DFishdf(filename)

DF       *filename    /* name of HDF file */
```

Purpose: To tell if a file is an HDF file.

Purpose: To write out the DD blocks necessary to update the file.

Returns: 0 on success; -1 on failure.

All data elements that have been written or partially written to the file are given valid DDs.

`DFupdate` is useful because HDF does not automatically write DDs to a file when the corresponding data is written out. Instead, it keeps all of the DDs in a special structure in primary memory, then writes them all out when `DFclose` is called.

If a crash occurs after data has been written out to a file but before the file is closed, changes that have been made to the file can be lost. If a `DFupdate` is performed immediately after a write, then the changes made by the write are not lost because both the DDs and the corresponding data have been stored in the file.

DFdup

FORTRAN:

```
INTEGER FUNCTION DFdup(dfile, tag, ref, otag, oref)

INTEGER  dfile          -   identifier of file access path
INTEGER  tag, ref       -   new tag and ref to point to old data
INTEGER  otag, oref     -   tag & ref that already point to data
```

C:

```
int DFdup(dfile, tag, ref, otag, oref)

DF       *dfile;      /* file access path */
uint16   tag, ref;    /* new tag & ref to point to old data */
uint16   otag,oref;   /* tag & ref that already point to data */
```

Purpose: To generate a DD whose offset and length are the same as those of another tag/ref.

Returns: 0 on success and a negative error indicator if `otag/oref` is not present in the file.

Sometimes it is desirable to have more than one DD point to a single data element. In such cases, the offset and length fields are identical for two or more DDs. `DFdup` is used to generate new references to data that are already referenced from somewhere else.

`DFdup` creates a new tag/ref (`tag/ref`) that points to the same data as an old tag/ref (`otag/oref`). If the new tag/ref combination already exists in the file, then the offset and length are changed to their new values.

WARNING: When a data element that is referenced from several places is modified, there is a danger that duplicate references to that data no longer point to the correct data. For instance, when a data element is moved, HDF does not automatically change all prior references to that data to point to the data in its new location.

DFwrite starts at the last position left by a DFaccess or DFwrite command and writes up to len bytes, then leaves the write pointer at the end of the element.

DFwrite fails if the DFaccess mode is not 'w' or 'a' ("w" or "a").

NOTE: The corresponding tag entry in the DD block is not updated until DFclose or DFupdate is called.

DFseek

FORTRAN:

```
INTEGER FUNCTION DFseek(dfile, offset)

INTEGER      dfile        -  identifier of file access path
INTEGER      offset       -  offset within data element to seek
                          to
```

C:

```
int32 DFseek(dfile, offset)

DF       *dfile;      /* file access path */
int32    offset;      /* offset within data to seek to */
```

Purpose: To set the read pointer to an offset within a data element.

Returns: On success, number of bytes from the beginning of the element to the new pointer position; on failure, negative.

DFseek tries to count offset number of bytes from the beginning of the element and set the read pointer there. The next time DFread is called, the read occurs from the new position.

If offset is such that the seek goes out of the range of the data item, an error is reported.

NOTE: DFseek is only valid when used in conjunction with a DFread. It should not be used to position a write operation.

Manipulating Data Descriptors (DDs)

These routines perform operations on DDs without doing anything with the data to which the DDs refer.

DFupdate

FORTRAN:

```
INTEGER FUNCTION DFupdate(dfile)

INTEGER      dfile      -   identifier of file access path
```

C:

```
int DFupdate(dfile)

DF *dfile;      /* file access path */
```

Purpose: To initiate a read or write on the data element with the specified tag/ref combination. For read, write, or append access, respectively, `access` is 'r', 'w', or 'a' (FORTRAN) and "r", "w", or "a" (C).

Returns: 0 on success; -1 on failure.

`DFaccess` must be invoked before the first `DFread` or `DFwrite` operation can be performed. It checks that the access mode is valid and moves to the first byte of the desired element in the file. If *append* access is specified, subsequent writes are appended to the end of the existing data.

DFread

FORTRAN:

```
INTEGER FUNCTION DFread(dfile, data, len)

INTEGER      dfile        - identifier of file access path
INTEGER      len          - number of bytes to read
CHARACTER*1  data(len)    - array that will hold data
```

C:

```
int DFread(dfile, data, len)

DF       *dfile;      /* file access path */
char     *data;       /* array that will hold data */
int32    len;         /* number of bytes to read */
```

Purpose: To read a portion of a data element.

Returns: If `DFread` is able to read any bytes, it returns the number of bytes read. If it is at the end of the element *before* the read occurs, it returns zero. On failure, it returns -1.

`DFread` starts at the last position left by a `DFaccess`, `DFread` or `DFseek` command and reads into the array `data`; i.e., any data that remains in the element up to `len` bytes. It fails if the `DFaccess` mode was not 'r' ("r").

DFwrite

FORTRAN:

```
INTEGER FUNCTION DFwrite(dfile, data, len)

INTEGER      dfile        - identifier of file access path
INTEGER      len          - number of bytes to write
CHARACTER*1  data(len)    - array with data to be written
```

C:

```
int DFwrite(dfile, data, len)

DF       *dfile;      /* file access path */
char     *data;       /* array with data to be written */
uint16   len;         /* number of bytes to wriTe */
```

Purpose: To append data to a data element.

Returns: Number of bytes written on success; negative on failure.

DFput (FORTRAN) and DFputelement (C)

FORTRAN:

```
INTEGER FUNCTION DFput(dfile, tag, ref, storage, len)

INTEGER       dfile           - identifier of file access path
INTEGER       tag, ref        - tag and ref of data to store
INTEGER       len             - number of bytes of data to store
CHARACTER*1   storage(len)    - array with data to be stored
```

C:

```
int DFputelement(dfile, tag, ref, ptr, len)

DF       *dfile;        /* file access path */
uint16   tag, ref;      /* tag and ref of data to be stored */
char     *ptr;          /* pointer to data to be stored */
int32    len;           /* number of bytes of data to be
                           stored */
```

Purpose: To add or replace elements in `dfile`.

Returns: If `DFputelement` succeeds, the return value is the number of bytes written. If it fails, -1 is returned.

The first `len` bytes in the array `storage` are written to the file with tag/ref referring to them.

NOTE: Since there can be no two elements with the same tag and reference numbers, any call with a tag/ref combination that duplicates an existing tag/ref replaces the previous element.

NOTE: The corresponding tag entry in the DD block is not updated until `DFclose` or `DFupdate` is called.

Reading or Writing Part of a Data Element

The second set of routines for reading and writing data elements makes it possible to read or write all or part of a data element, in contrast to `DFput` and `DFget`, described in the previous section, which can only read or write entire elements.

DFaccess

FORTRAN:

```
INTEGER FUNCTION DFaccess(dfile, tag, ref, access)

INTEGER      dfile        -  identifier of file access path
INTEGER      tag, ref     -  tag and ref of data to be accessed
CHARACTER*1  access       -  access mode
```

C:

```
int DFaccess(dfile, tag, ref, access)

DF       *dfile;      /* file access path */
uint16   tag, ref;    /* tag and ref of data to access */
char     *access;     /* access mode */
```

Returns: 0 on success; if there are no more elements which match the pattern, a negative number is returned and `tag` and `ref` are set to zero.

The C version of this program returns a pointer to the actual data descriptor for the tag/ref that have be set. Since FORTRAN cannot, in general, support pointers or structures, the FORTRAN version returns in three separate variables the tag, ref, and length from the data descriptor.

In addition to the values that are returned, `DFfind` updates the file structure pointed to by `dfile` to indicate that the data object that has been located is the "current" one.

If `DFsetfind` has set specific tag and ref values, `DFfind` prepares the HDF system to access the corresponding object in the file. If `ref` were the flag `DFREF_WILDCARD`, then `DFfind` prepares the system to access the next ref for the specified tag. Similarly, if the tag was the flag `DFREF_WILDCARD`, then `DFfind` prepares the system to access the next tag for the specified ref.

Storing and Retrieving Entire Data Elements

There are two sets of routines for reading and writing data elements. `DFput (DFputelement)` and `DFget (DFgetelement)`, described here, store and retrieve *entire* elements. A second set of routines may be used if access to only *part* of a data element is desired. These routines are covered in the next section.

DFget (FORTRAN) and DFgetelement (C)

FORTRAN:

```
INTEGER FUNCTION DFget(dfile, tag, ref, storage)

INTEGER        dfile      -   identifier of file access path
INTEGER        tag, ref   -   tag and ref of data to be
                              retrieved
CHARACTER*1    storage(*) -   array for storing incoming data
```

C:

```
int DFgetelement(dfile, tag, ref, ptr)

DF         *dfile;    /* file access path */
uint16     tag, ref;  /* tag and ref of data to be retrieved */
char       *ptr;      /* pointer to space for storing
                         incoming data */
```

Purpose: To extract the data referred to by the tag/ref and place the data in the array `storage`.

Returns: If `DFget` succeeds, the return value is the number of bytes read. If it fails, -1 is returned.

DFsetfind and DFfind provide a kind of "seeking" capability, where the objective is to seek to a particular data descriptor in the data descriptor block, rather than to seek to a byte offset in the file.

DFsfind (FORTRAN) and DFsetfind(C)

FORTRAN:

```
INTEGER FUNCTION DFsfind(dfile, tag, ref)

INTEGER      dfile        -  identifier of file access path
INTEGER      tag, ref     -  tag and ref to be located
```

C:

```
int DFsetfind(dfile, tag, ref)

DF *dfile;      /* file access path */
uint16 tag, ref;      /* tag and ref to be located */
```

Purpose: To initialize searches for elements using tags or reference numbers.

Returns: 0 on success and -1 on failure.

DFsetfind initializes any search for elements with a given tag or ref by setting tag and ref values in the file pointer, dfile.

The parameter ref may be replaced by the "wildcard" flag, DFREF_WILDCARD, which indicates that the search by DFfind is to begin with the first ref in the file that matches the tag. Subsequent calls to DFfind return the refs for the specified tag in ascending order.

Similarly, the tag parameter may be DFTAG_WILDCARD, indicating that DFfind is to begin with the first tag that occurs with the specified ref.

If both tag and ref are wildcards, DFfind will return all of the tags and reference numbers in the file in ascending order with reference numbers as the primary field.

DFfind

FORTRAN:

```
INTEGER FUNCTION DFfind(dfile, tag, ref, len)

INTEGER      dfile         -  identifier of file access path
INTEGER      tag, ref      -  tag/ref of data
INTEGER      len           -  number of bytes of data
```

C:

```
int DFfind(dfile, ptr)

DF *dfile;            /* file access path */
struct DFdesc *ptr    /* pointer to data descriptor for data */
```

Purpose: To locate the data descriptor needed for the next read from the file.

Purpose: To update the DD blocks, then close the access path to the file referred to by `dfile`.

Returns: 0 on success and -1 on failure.

NOTE: If the contents of a file have been changed, it is important to call `DFclose` to ensure that the DD blocks are written to the file. (To cause all DD blocks to be written to the file without also closing the file, see `DFupdate`.)

Finding the Next Occurrence of a Given Object

DFfindnextref

FORTRAN:

```
INTEGER FUNCTION DFfindnextref(dfile, tag, lref)

INTEGER     dfile       - identifier of file access path
INTEGER     tag         - tag to look for
INTEGER     ref         - ref after which to search
```

C:

```
int DFfindnextref(dfile, tag, lref)

DF       *dfile;      /* file access path */
int16    tag,         /* tag to look for */
         lref;        /* ref after which to search */
```

Purpose: To find the reference number for the next occurrence after the last ref of the given tag.

Returns: The desired ref on success; -1 on failure.

This routine is useful for stepping through a file one object at a time. For instance, it systematically reads all the annotations associated with all occurrences of a given HDF object.

Finding Data Descriptors for HDF Objects

The routines *DFsetfind* and *DFfind* are for locating data descriptors in a file. Given the tag and ref of a data object, `DFsetfind` initializes a search for the object's data descriptor, and `DFfind` returns information from the data descriptor about the data. This information is necessary for accessing the actual data.

`DFsetfind` and `DFfind` also make it possible to locate elements without previous knowledge of their tags or reference numbers (or both). For example, if you know that several instances of a given tag are in a file, but do not know the reference numbers, you can use these routines to find the reference numbers and their corresponding data descriptors. One call to `DFsetfind`, followed by successive calls to `DFfind`, lets you step through all of the refs that go with the tag.

macro processors `m4` and `cpp` let your compiler include and preprocess header files. If this or a similar capability is not available, you may have to copy whatever declarations, definitions, or values you need from `dfF.h` into your program code.

Opening and Closing Files

DFopen

FORTRAN:

```
INTEGER FUNCTION DFopen(filename, access, defDDs)

CHARACTER*64 filename   -    name of file to be opened
INTEGER access          -    type of access
INTEGER defDDs          -    number of data descriptors to
                             allocate per block
```

C:

```
DF *DFopen(filename, access, defDDs)

char    *filename;     /* name of file to be opened */
int     access; /* type of access */
int     defDDs; /* number of data descriptors to allocate
per block*/
```

Purpose: To provide an access path to the file named in `filename` with the access given in `access`.

Returns: An *integer* (FORTRAN) or *pointer* (C) that identifies the file access path. If the call succeeds, the return value is positive. If the call fails, the return value is 0 (FORTRAN) or NULL (C).

`DFopen` also reads into memory all of the DD blocks in the file.

Values allowed for `access` are `DFACC_READ` for read only, `DFACC_WRITE` for write only, `DFACC_ALL` for read and write, and `DFACC_CREATE` for create or overwrite. If the file must be created or extended, `defDDs` specifies the number of data descriptors (DDs) to be allocated per DD block. If `defDDs` ≤ 0, the number of data descriptors is set to the machine default.

NOTE: In the current implementation, only one file can be open at a time.

DFclose

FORTRAN:

```
INTEGER FUNCTION DFclose(dfile)

INTEGERdfile   -      identifier of file access path
```

C:

```
int DFclose(dfile)

DF *dfile;      /* file access path */
```

Table 6.1 General Purpose Routines in the HDF Library(Continued)

Long Name	Short Name	Function
DFgetelement	**dfget**	extracts the data referred to by the tag/ref and places the data in the array storage.
DFputelement	**dfput**	adds or replaces elements in dfile.
DFaccess		inititiates a read or write on the data element with the specified tag/ref combination.
DFread		reads a portion of a data element.
DFwrite		appends data to a data element.
DFseek		sets the read pointer to an offset within a data element.
DFupdate		writes out the DD blocks necessary to update the file.
DFdup		generates a DD whose offset and length are the same as those of another tag/ref. I.e. duplicates a DD.
DFdel		deletes a tag/ref from the list of DDs.
DFerrno		reports the value of DFerror.
DFishdf		tells if a file is an HDF file.
DFnewref		generates an unused reference number.
DFnumber		counts the number of occurrences of a given tag in the HDF file.
DFstat		provides status information about an HDF file.

Header Files

You may need to include a header file if your program uses special HDF declarations or definitions. There are two header files, one for FORTRAN and one for C, that apply to the general purpose I/O routines listed here:

- *dfF.h* contains the declarations and definitions that are used by FORTRAN routines.
- *df.h* contains the declarations and definitions that are used by C routines.

For example, if your program uses mnemonics for tags, the corresponding numerical values for the tags can be found in `dfF.h` (FORTRAN) or `df.h` (C). The contents of `dfF.h` and `df.h` are listed in Appendix B.

Although the use of header files is always permitted in C programs, it is generally not permitted in FORTRAN. It is, however, sometimes available as an option in FORTRAN. For example, on UNIX systems, the

Chapter Overview

The following chapter is a reference to the set of general purpose low-level routines used when working with HDF files. Note that if you are using a higher level HDF interface, such as RIS8, you probably do not need to use these low-level routines.

Introduction

At times you may need a lower level of access to HDF files than that provided by more application-oriented interfaces such as RIS8 and SDS. The routines described in this chapter enable you to build and manipulate HDF files of any type, including those of your own invention. All HDF applications developed at NCSA use these routines as their basic building blocks.

NOTE: If you are using an HDF package such as RIS8, you probably do not need to use any of these routines. In fact if you use some of these lower level routines while simultaneously accessing a file with the higher level interfaces, the higher level routines may not function properly.

In order to understand how these routines work, it is important to understand the underlying structure of all HDF files. Detailed information about the basic HDF structure and how it works can be found in *NCSA HDF Specifications*, which may be obtained by contacting NCSA's Software Tools Group at the address listed on the README page at the beginning of this manual.

Table 6.1 lists the long and short names and the functions of the general purpose routines currently contained in the HDF library. The following sections provide descriptions and examples of these calling routines.

Table 6.1 General Purpose Routines in the HDF Library

Long Name	Short Name	Function
`DFopen`		provides an access path to the file named in filename with the access given in access.
`DFclose`		updates the DD blocks, then closes the access path to the file referred to by dfile.
`DFfindnextref`	`dfindnr`	finds the ref number for the next occurrence of a given tag.
`DFsetfind`	`dfsfind`	initializes searches for elements using tags or reference numbers.
`DFfind`	`dffind`	locates the data descriptor needed for the next read from the file.

Chapter 6 General Purpose HDF Routines