Chapter 3

The Bare Necessities for Reading and Writing Data

3.0 Chapter Overview

Each of the six data models supported by HDF provides an interface for reading, writing and organizing data of a particular type. In Chapter 4 you will learn to perform a simple read and write operation for each type of data using the "add" and "get" functions available in each model. Each of the six sections in this chapter is dedicated to a different data model. As a result, each section introduces one data model, describes its interface and provides both a C and FORTRAN example for reading and writing data using the appropriate function calls.

In an effort to promote simplicity, most of the code examples presented in Chapter 4 use the minimum number of function calls necessary to store and retrieve data from an HDF file. In most cases, these examples take advantage of HDF default parameters to simplify the process of reading and writing data. However, there are a few examples that include optional function calls in order to demonstrate how to override default settings. All optional calls are clearly marked as such.

In Chapter 4 you will learn to perform the following operations:

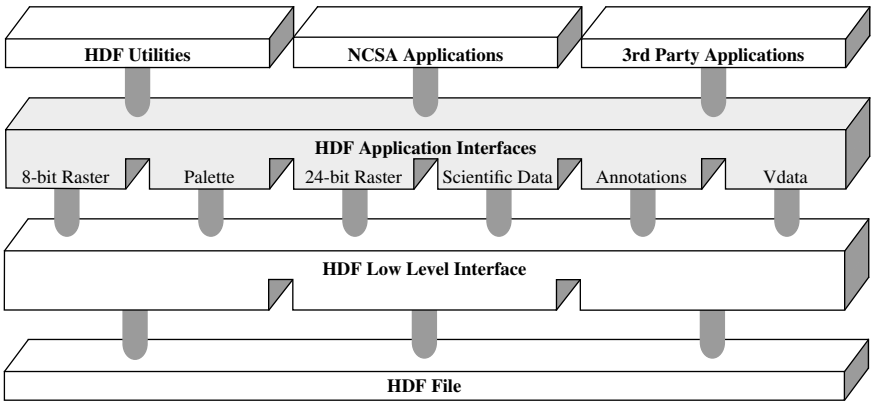
- The ***8-bit raster model***: Write and read an 8-bit raster image.
- The ***palette model***: Write and read an 8-bit palette outside of a raster image set.
- The ***24-bit raster model*** Write and read a 24-bit image using scan-plane interlacing.
- The ***scientific data model*** Write and read a 3-dimensional array 64-bit floats.
- The ***annotation model***: Write and read a file description.
- The ***Vdata model:*** Write and read 32-bit floating-point records.

After reading this chapter, you will know enough about HDF to begin creating simple files. In fact, after you complete this material you can use the data handling techniques introduced here without learning another thing. Of course, in order to master more sophisticated data hierarchies you'll need to continue working your way through this manual and the HDF User's Guide.

3.1 Introducing the Application Interfaces Supported by HDF

There are currently six different application interfaces available in the HDF library (See FIGURE 3.1a). Each interfaces provides a collection of functions for reading, writing, and organizing one type of data and its associated data elements.

FIGURE 3.1a **HDF supports six different application interfaces.**



In general, the more flexible the interface, the more "set-up" or "query" work you must do before reading or writing data (See TABLE 3A). The 8-bit raster data model, for example, is extremely well-defined, therefore no set-up work is required before reading or writing 8-bit raster data. Both operations are handled through one function call assuming the image dimensions are known. On the other hand, the most flexible, and least-structured interface is the Vdata interface. As a result, reading or writing Vdata takes a bit more set-up and query work than any other type of data.

TABLE 3A **Some function calls available for reading and writing data via the different HDF data models.**

Operation		8-bit Raster	Palette	24-bit Raster	Scientific Data	Annotations	Vdata
Writing	**Set-up**	-	-	-	DFSDsetNT*	-	VSattach VSsetname* VSfdefine VSsetfields VSdetach
	Write	DFR8addimage	DFPaddpal	DF24addimage	DFSDadddata	DFANaddfds	VSwrite
Reading	**Query**	DFR8getdims*		DF24getdims*	DFSDgetdims* DFSDgetNT*	DFANgetfdslen	VSgetid* VSattach VSinquire VSsetfields VSdetach
	Read	DFR8getimage	DFPgetpal	DF24getimage	DFSDgetdata	DFANgetfds	VSread

*indicates optional function call

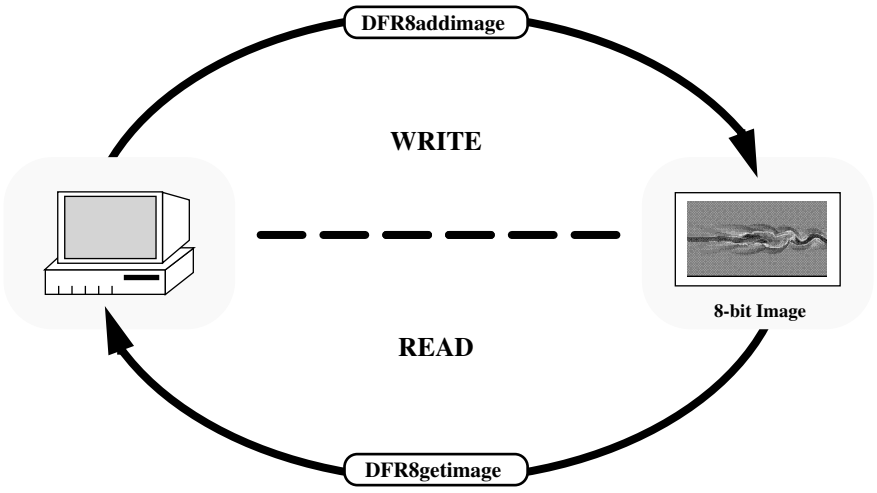
Every application interface in the HDF library is supported by an independent set of function calls. Each set of function calls is assigned an ***interface prefix*** which identifies the interface to which it belongs. In most cases the prefix is an abbreviation for the data model supported by the interface. For example, all 8-bit raster functions use the prefix DFR8; an abbreviation for "data format, raster 8." The palette model uses DFP, for "data format, palette" and the scientific data model uses DFSD for "data format, scientific data." Once you are familiar with the six different interfaces, identifying a function call by it prefix becomes quite simple.

3.2 HDF and the 8-bit Raster Interface

Every function in the 8-bit raster interface begins with the prefix ***DFR8*** which stands for "data format, raster 8." By using DFR8 function calls, HDF can make a series of assumptions about the raster operations you are performing. These assumptions, or ***default settings***, reduce the amount of work you must do to perform a successful read or write operation.

When you make a DFR8 call, HDF assumes that you are interested in reading or writing data as a two-dimensional array of 8-bit integers. Consequently, reading or writing an 8-bit raster image to file requires only one function call (See FIGURE 3.2a). To write a raster to a file, you can use **DFR8addimage**. To read a raster from a file, you must use **DFR8getimage**.

FIGURE 3.2a **8-bit image data can be read or written to a file via a single read or write function call.**



Although **DFR8addimage** and **DFR8getimage** are the only two calls required for reading and writing 8-bit raster data, the DFR8 interface provides several additional functions for more advanced operations. For more information on additional DFR8 function calls, consult the HDF User's Guide.

Writing an 8-bit Raster Image to an HDF File

If you are programming in C, **DFR8addimage** is the only function call you need to write a raster image to file**.** In FORTRAN, the equivalent function is called **d8aimg**. To write an 8-bit raster, the call made from your program should look something like this:

```
C:        DFR8addimage (filename, image, width, height, compress)
FORTRAN:  d8aimg (filename, image, width, height, compress)
```

In either case, the "add image" call appends an image to an existing HDF file or creates a new file if the named file does not exist. It does not overwrite data in existing files. The parameters for both the C and FORTRAN call are more explicitly defined below (See TABLE 3B on page 20).

TABLE 3B **The parameters required for DFR8addimage.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DFR8addimage (d8aimg)	filename	character *(*)	character pointer	Name of file to store the image data.
	image	character array	character array	Array containing the image data.
	width	integer	32-bit integer	Number of columns in the image.
	height	integer	32-bit integer	Number of rows in the image.
	compress	integer	integer	Type of compression: (none=0, run length=DFTAG_RLE).

Notice in the following example, that the order in which the dimensions for the image array are declared differs between C and FORTRAN. To efficiently read and write raster data, the FORTRAN declaration requires the width (rows) before the height (columns), whereas the C declaration requires the height before the width.

EXAMPLE 1. **Writing an 8-bit raster image to an HDF file.**

In the following code example, **DFR8addimage** and **d8aimg** are used to write an 8-bit image to a file called “example.hdf.” In both the C and FORTRAN example, the pixel values for the raster image are initialized using the following values:

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25
26	27	28	29	30

C:

```
#include "hdf.h"

#define WIDTH 5
#define HEIGHT 6

main()
{

    /* Initialize the image array */
    static uint8 raster_data[HEIGHT][WIDTH] =

                                          { 1,  2,  3,  4,  5,
                                            6,  7,  8,  9, 10,
                                           11, 12, 13, 14, 15,
                                           16, 17, 18, 19, 20,
                                           21, 22, 23, 24, 25,
                                           26, 27, 28, 29, 30 };


    /* Write the 8-bit raster image to file */
    DFR8addimage("example.hdf", raster_data, WIDTH, HEIGHT, 0);
}
```

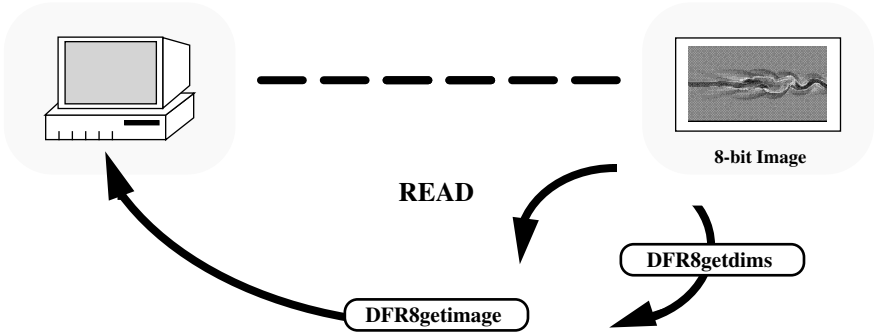
FORTRAN:

```
      PROGRAM RASTER8

      character*1 raster_data(5,6)
      integer istat, d8aimg

C Initialize the image array
      data raster_data /  1,  2,  3,  4,  5,
     $                    6,  7,  8,  9, 10,
     $                   11, 12, 13, 14, 15,
     $                   16, 17, 18, 19, 20,
     $                   21, 22, 23, 24, 25,
     $                   26, 27, 28, 29, 30 /

C Write the 8-bit raster image to the file
      istat = d8aimg('example.hdf', raster_data, 5, 6, 0)

      stop
      end
```

Reading an 8-bit Raster Image from an HDF File

If you are programming in C, **DFR8getimage** is the only function call you need to read a raster image from a file**.** In FORTRAN, the equivalent function is called **d8gimg**. If you are not sure of the dimensions of an image you intend to read, the **DFR8getdims** function (or **d8gdims** in FORTRAN) can obtain them for you (See FIGURE 3.2b). **DFR8getdims** is optional. However, by using **DFR8getdims** in your program, you can verify the size of the space required to hold the image before reading any data.

FIGURE 3.2b **DFR8getdims can be used before DFR8getimage to obtain the dimensions of an image.**



If you already know the dimensions of the image you intend to read, the function call made from your program should look something like this:

C:
```
DFR8getimage (filename, image, width, height, palette)
```
FORTRAN:
```
d8gimg (filename, image, width, height, palette)
```

However, if you wish to verify the dimensions before reading the image, your program should include the following calls:

C:
```
DFR8getdims (filename, width, height, haspalette)
DFR8getimage (filename, image, width, height, palette)
```

FORTRAN:
```
d8gdims (filename, width, height, haspalette)
d8gimg (filename, image, width, height, palette)
```

The first time the get dimensions call is made, it returns information about the first 8-bit raster image in the file. Any subsequent calls will return dimensions for successive images in the file. The “get image” call operates in a similar manner. The parameters for both the “get dimensions” and the “get image” calls are more explicitly defined below(See TABLE 3C).

TABLE 3C **The parameters required for DFR8getdims and DFR8getimage.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DFR8getdims (d8gdims)	filename	character *(*)	character pointer	Name of file containing the image data.
	width	integer	32-bit integer pointer	Number of columns.
	height	integer	32-bit integer pointer	Number of rows.
	haspalette	integer	integer pointer	Reports on palette (yes=1, no=0).
DFR8getimage (d8gimg)	filename	character *(*)	character pointer	Name of file containing the image data.
	image	character array	character array	Array to hold the image data.
	width	integer	32-bit integer	Width of the buffer for the image data.
	height	integer	32-bit integer	Height of the buffer for the image data.
	palette	character array	character array	Array to hold the palette data.

Notice in the following example, as in the case of **DFR8addimage**, that the order in which the dimensions for the image array are declared differs between C and FORTRAN. The FORTRAN declaration requires the width before the height, whereas the C declaration requires the height before the width.

EXAMPLE 2. **Reading an 8-bit Raster Image from an HDF file**

The following example will query a file called “example.hdf” for the dimensions of an 8-bit image. Although the **DFR8getdims** call is optional, it is included as a demonstration of how to check the dimensions of an image before reading its image data. If the dimensions are already known, only the **DFR8getimage** call is required. This example also assumes that no palette is grouped with the image data, therefore NULL is passed as the palette parameter. Any time the palette argument is NULL (or 0 in the case of FORTRAN), all palette data is ignored.

C:

```
#include "hdf.h"

#define WIDTH 5
#define HEIGHT 6

main()
{
    uint8 raster_data[HEIGHT][WIDTH];
    int32 width, height;
    intn haspal;

    /* Get the dimensions of the image */
    DFR8getdims("example.hdf", &width, &height, &haspal);

    /* Read the raster data if the dimensions are correct */
    if(width<=WIDTH && height<=HEIGHT)
        DFR8getimage("example.hdf",raster_data,width,height,NULL);
}
```

FORTRAN:

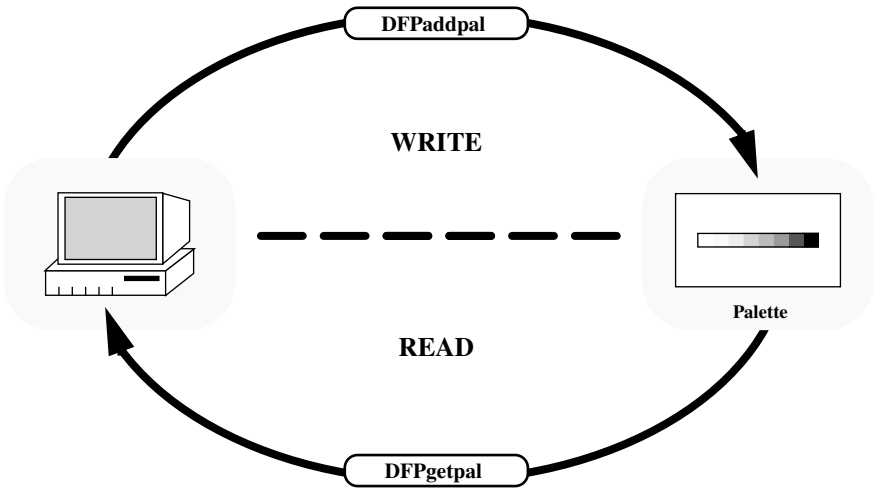
```
      PROGRAM RASTER8

      character*1 image(5,6)
      integer istat, height, width, d8gimg, haspal

C     Get the dimensions of the image
      istat = d8gdims('example.hdf', width, height, haspal)

C     Read the raster data if the dimensions are correct
      if ( (width .le. 6) .and. (height .le. 5) ) then
         istat = d8gimg('example.hdf', image, width, height, 0)
      endif

      stop
      end
```

3.3 HDF and the Palette Interface

Every function in the palette interface begins with the prefix ***DFP***, which stands for "data format, palette." By using DFP function calls, HDF can make a series of assumptions about the operations you are performing. These assumptions reduce the amount of work you must do to perform a successful read or write operation.

When you make a DFP call, HDF assumes you are interested in reading or writing data as a 768-byte array of 8-bit integers. HDF also assumes that the first 256-bytes in the array contain the red components, the second 256-bytes contain the green components, and that the last 256-bytes contain the blue components. Based on these assumptions, reading or writing a palette to an HDF file requires only one function call (See FIGURE 3.3a). To write a palette to a file, you can use the **DFPaddpal** function call. To read a palette from a file you must use **DFPgetpal**.

FIGURE 3.3a **Palettes can be read or written to a file via a single read or write function call.**



Although **DFPaddpal** and **DFPgetpal** are the only calls required for reading and writing palettes, the DFP interface provides several additional functions for more advanced palette operations. For more information on additional DFP function calls, consult the HDF User's Guide.

Writing a Palette to an HDF File

If you are programming in C, **DFPaddpal** is the only function call you need to write a palette to file**.** In FORTRAN, the equivalent function is called **dpapal**. To write a palette via the DFP interface, your program should contain the following function call:

```
C:          DFPaddpal (filename, palette)
FORTRAN:    dpapal (filename, palette)
```

In either case, the "add palette" call appends a palette to an existing HDF file or create a new one if the named file does not exist. It does not overwrite data in existing files. The parameters for both calls are more explicitly defined below (See TABLE 3D on page 25).

TABLE 3D **The parameters required for DFPaddpal.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DFPaddpal (dpapal)	filename	character *(*)	character pointer	Name of file that will store the palette.
	palette	character array	character array	Array holding the palette.

Calling **DFPaddpal** immediately after writing an 8-bit raster image will not explicitly group the palette with the preceding image. Palettes written to file sequentially can be retrieved sequentially. However, to maintain a higher level of organization between multiple palettes and images stored in the same file, it's often better to explicitly group each palette with the image to which it belongs. To find out more about assigning a palette to an image, see the HDF User's Guide.

EXAMPLE 3. **Writing a palette to an HDF file**

In the following code example, **DFPaddpal** and **dpapal** are used to write a palette to a file called "example.hdf." In both the C and FORTRAN example, the palette data is initialized to the following grayscale values:

Palette Entry
0
1
. . .
255

Red Component	Green Component	Blue Component
0	0	0
1	1	1
. . .	. . .	. . .
255	255	255

C:

```
#include "hdf.h"

main()
{
    uint8 palette_data[768];
    intn i;

/* Initialize the palette to grayscale */
for(i=0; i<256; i++) {
    palette_data[i*3]=i;
    palette_data[i*3+1]=i;
    palette_data[i*3+2]=i;
  }

/* Write the palette to file */
DFPaddpal("example.hdf",palette_data);
}
```

FORTRAN:

```
      PROGRAM PALETTE

      INTEGER         dpapal, ret, i
      CHARACTER*1     palette_data(768)

C Initialize the palette to grayscale
      do 10, i=1,256
         palette_data((i-1)*3+1) = char(i-1)
         palette_data((i-1)*3+2) = char(i-1)
         palette_data((i-1)*3+3) = char(i-1)
   10 continue

C Write the palette to file
      ret = dpapal('example.hdf', palette_data)

      stop
      end
```

Reading a Palette from an HDF File

If you are programming in C, **DFPgetpal** is the only function call you need to read a palette from a file**.** In FORTRAN, the equivalent function is called **dpgpal**. To read a palette, the function call made from your program should look something like this:

```
C:        DFPgetpal (filename, palette)
FORTRAN:  dpgpal (filename, palette)
```

The first time a “get palette” call is made, it returns the first palette in the file. Repeatedly calling get palette returns successive palettes in the file. Successive get palette calls made via the DFP interface will return all palettes in a file, even those stored via the DFR8 interface. The parameters for the “get palette” call are more explicitly defined below (See TABLE 3E).

TABLE 3E **The parameters required for DFPgetpal.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DFPgetpal (dpgpal)	filename	character*(*)	character pointer	Name of file storing the palette.
	palette	character array	character array	Buffer to hold the palette.

EXAMPLE 4. **Reading a Palette from an HDF file**

The following example will read a palette from an HDF called “example.hdf.”

C:

```
#include "hdf.h"

main()
{
    uint8 palette_data[768];

/* Read the palette data from a file */
DFPgetpal("example.hdf",palette_data);
}
```

FORTRAN:

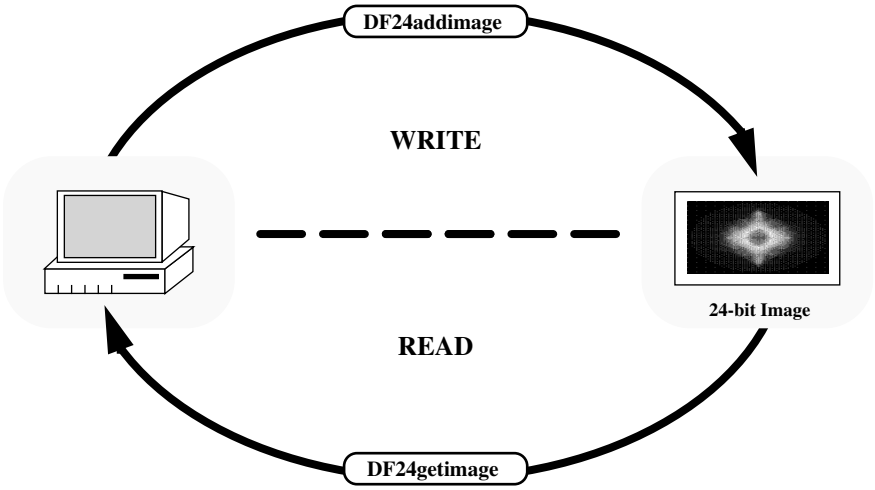
```
      PROGRAM PALETTE

      INTEGER         dpgpal
      CHARACTER*1     palette_data(768)

C Read the palette from a file
      ret = dpgpal('example.hdf', palette_data)

      stop
      end
```

3.4 HDF and the 24-bit Raster Image Interface

Every function in the 24-bit raster interface begins with the prefix ***DF24*** which stands for "data format, 24-bit image." By using DF24 function calls, HDF can make a series of assumptions about the raster operations you are performing. These assumptions reduce the amount of work you must do to perform a successful read or write operation.

When you make a DF24 call, HDF assumes that you are interested in reading or writing data as a three-dimensional array of 8-bit integers. HDF also assumes that you are reading or writing the image data using pixel interlacing. These assumptions make it possible for you to read or write 24-bit images using only one function call (See FIGURE 3.4a). To write an image to a file, you can use **DF24addimage**. To read a raster from a file, you must use **DF24getimage**.

FIGURE 3.4a **24-bit images can be read or written to a file via one read or write function call.**

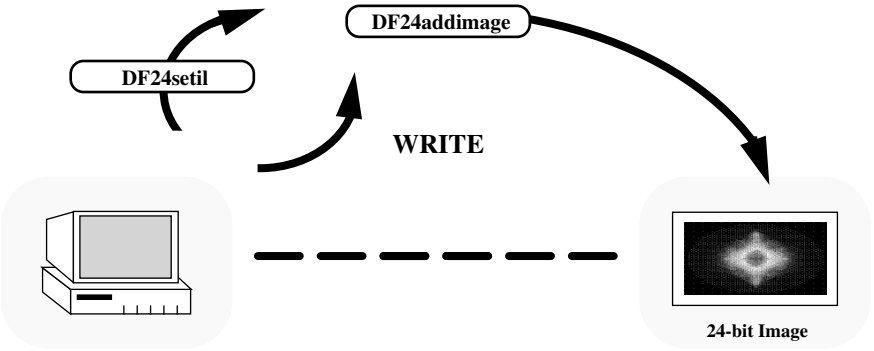


Although **DF24addimage** and **DF24getimage** are the only calls required for reading or writing 24-bit raster data to file, the DF24 interface provides a host of optional function for accessing data through more advanced operations. For more information on additional DF24 function calls, consult the HDF User's Guide.

Writing a 24-bit Image to an HDF File

If you are programming in C, **DF24addimage** is the only function call you need to write a 24-bit image to file. In FORTRAN, the equivalent function is called **d2aimg**. In either case, the default setting is pixel interlacing. If you prefer to store the image using scan-line interlacing or scan-plane interlacing, you must call **DF24setil** (or **d2setil** in FORTRAN) to change the interlacing scheme (See Figure 3.4b on page 28).

FIGURE 3.4b **DF24setil can be used to override the default pixel interlace setting.**



If you wish to write a 24-bit raster image to file using pixel interlacing, the function call made from your program should look something like this:

C: `DF24addimage (filename, image, width, height)`
FORTRAN: `d2aimg (filename, image, width, height)`

If you wish to change the interlacing scheme, your program should include the following calls:

C:
```
DF24setil (interlacing)
DF24addimage (filename, image, width, height)
```

FORTRAN:
```
d2setil (interlacing)
d2aimg (filename, image, width, height)
```

The parameters for both the “set interlace” and “get image” calls are more explicitly defined below (See TABLE 3F). Notice that passing DFIL_LINE via the interlacing argument in **DF24setil**, changes the pixel interlacing scheme to scan-line interlacing. In the same sense, passing DFIL_PLANE changes the interlacing to scan-plane interlacing.

TABLE 3F **The parameters required for DF24setil and DF24addimage.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DF24setil (d2setil)	interlacing	integer	integer	Specifies the type of interlacing to use. DFIL_PIXEL = pixel, DFIL_LINE = scan-line DFIL_PLANE =scan-plan
DF24addimage (d2aimg)	filename	character *(*)	character pointer	Name of file that will store the image.
	image	character array	character array	Array containing the image data.
	width	integer	32-bit integer	Number of columns in the image.
	height	integer	32-bit integer	Number of rows in the image.

EXAMPLE 5.

Writing a 24-bit Raster Image to an HDF File

In the following example, **DF24addimage** and **d2aimg** are used to write a 24-bit image to a file called "example.hdf." Although the **DF24setil** function call is optional, it is included here to demonstrate how to change the default interlace setting from pixel interlacing to scan-plane interlacing. Consequently in both the C and FORTRAN example, the pixels for the image are initialized using the following values:

	1	2	3	4	5		
	6	7	8	9	10		
	11	1	2	3	4	5	
	16	6	7	8	9	10	
	21	11	1	2	3	4	5
Red Plane	26	16	6	7	8	9	10
		21	11	12	13	14	15
Green Plane		26	16	17	18	19	20
			21	22	23	24	25
Blue Plane			26	27	28	29	30

C:

```
#include "hdf.h"

#define WIDTH 5
#define HEIGHT 6

main()
{
    /* Initialize the image array */
    static uint8 raster_data[3][HEIGHT][WIDTH] =

                                          { 1,  2,  3,  4,  5,
                                            6,  7,  8,  9, 10,
                                           11, 12, 13, 14, 15,
                                           16, 17, 18, 19, 20,
                                           21, 22, 23, 24, 25,
                                           26, 27, 28, 29, 30,

                                            1,  2,  3,  4,  5,
                                            6,  7,  8,  9, 10,
                                           11, 12, 13, 14, 15,
                                           16, 17, 18, 19, 20,
                                           21, 22, 23, 24, 25,
                                           26, 27, 28, 29, 30,

                                            1,  2,  3,  4,  5,
                                            6,  7,  8,  9, 10,
                                           11, 12, 13, 14, 15,
                                           16, 17, 18, 19, 20,
                                           21, 22, 23, 24, 25,
                                           26, 27, 28, 29, 30  };

    /* Change interlace from pixel to scan-plane */
    DF24setil(DFIL_PLANE)

    /* Write the 24-bit image data to file */
    DF24addimage("example.hdf", raster_data, WIDTH, HEIGHT);
}
```

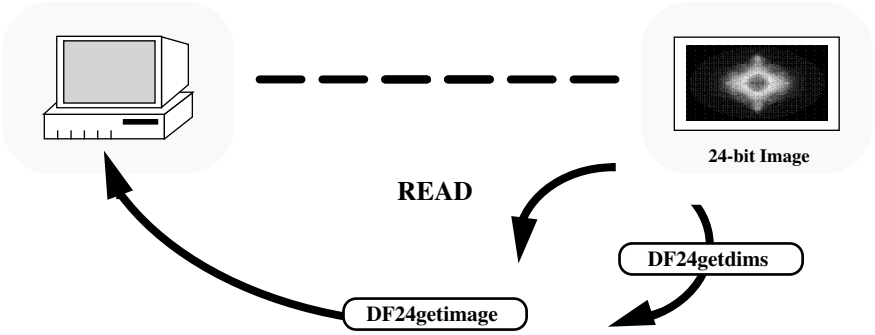
FORTRAN:

```
      PROGRAM RASTER24

      INTEGER         d2aimg, d2setil, ret, width, height
      CHARACTER*1     raster_data(5,6,3)
      DATA            width/5/, height/6/

C Initialize the image array
      DATA            raster_data /  1,  2,  3,  4,  5,
     $                               6,  7,  8,  9, 10,
     $                              11, 12, 13, 14, 15,
     $                              16, 17, 18, 19, 20,
     $                              21, 22, 23, 24, 25,
     $                              26, 27, 28, 29, 30,
     $
     $                               1,  2,  3,  4,  5,
     $                               6,  7,  8,  9, 10,
     $                              11, 12, 13, 14, 15,
     $                              16, 17, 18, 19, 20,
     $                              21, 22, 23, 24, 25,
     $                              26, 27, 28, 29, 30,
     $
     $                               1,  2,  3,  4,  5,
     $                               6,  7,  8,  9, 10,
     $                              11, 12, 13, 14, 15,
     $                              16, 17, 18, 19, 20,
     $                              21, 22, 23, 24, 25,
     $                              26, 27, 28, 29, 30 /

C Change interlace from pixel to scan plane
      ret = d2setil(2)

C Write the 24-bit image data to file
      ret = d2aimg('example.hdf', raster_data, width, height)

      stop
      end
```

Reading a 24-bit Raster Image from an HDF File

If you are programming in C, **DF24getimage** is the only function call you need to read a 24-bit raster image from a file**.** In FORTRAN, the equivalent function is called **d2gimg**. If you are not sure of the dimensions for an image you are attempting to read, the **DF24getdims** function (or **d2gdims** in FORTRAN) can obtain them for you (See FIGURE 3.4c). **DF24getdims** is optional. However, by using **DF24getdims** in your code, you can verify the size of the space required to hold the image data before reading any data. **DF24getdims** will also verify the interlacing scheme.

FIGURE 3.4c **DF24getdims can be used before DF24getimage to obtain the dimensions of an image.**



If you know the dimensions for the image you intend to read, the function call made from your program should look something like this:

```
C:          DF24getimage (filename, image, width, height)
FORTRAN:    d2gimg (filename, image, width, height)
```

However, if you wish to verify the dimensions before reading the image, your program should include the following calls:

```
C:          DF24getdims (filename, width, height, interlace)
            DF24getimage (filename, image, width, height)

FORTRAN:    d2gdims (filename, width, height, interlace)
            d2gimg (filename, image, width, height)
```

The first time a “get dims” call is made, it returns information about the first 24-bit raster image in the file. Repeatedly calling get dims returns information regarding successive 24-bit images in the file. “Get image” calls operate in a similar manner. Parameters for both the get dims and the get image call are more explicitly defined below (See TABLE 3G).

TABLE 3G **The parameters required for DF24getdims and DF24addimage.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DF24getdims (d2gdims)	filename	character *(*)	character pointer	Name of file containing the image data.
	width	integer	32-bit integer pointer	Number of columns in the image.
	height	integer	32-bit integer pointer	Number of rows in the image.
	interlace	integer	integer pointer	Type of color component interlacing
DF24getimage (d2gimg)	filename	character *(*)	character pointer	Name of file containing the image data.
	image	character array	character array	Buffer to hold the image data.
	width	integer	32-bit integer	Width of the image data buffer.
	height	integer	32-bit integer	Height of the image data buffer.

In the following example, as in the case of **DFR8addimage**, the order in which the dimensions for the image array are declared differs between C and FORTRAN. To efficiently read or write raster data, the FORTRAN declaration requires the width before the height, whereas the C declaration requires the height before the width.

EXAMPLE 6. **Reading a 24-bit Raster Image from an HDF file**

The following example queries an HDF file called “example.hdf” for the dimensions and interlacing scheme of a 24-bit image. Although the **DF24getdims** function call is optional, it is included in the following example as a demonstration of how to check the image dimensions and interlacing before reading any image data. If the image dimensions and interlacing scheme are already known, only the **DF24getimage** call is required.

C:

```
#include "hdf.h"

#define WIDTH 5
#define HEIGHT 6

main()
{
    uint8 raster_data[3][HEIGHT][WIDTH];
    int32 width, height;
    intn interlace;

    /* Get the image dimensions from a file */
    DF24getdims("example.hdf",&width, &height, &interlace);

    /* Read raster data if the dimensions and interlace are correct */
    if(width<=WIDTH && height<=HEIGHT && interlace==DFIL_PLANE)
        DF24getimage("example.hdf", raster_data, width, height);
}
```

FORTRAN:

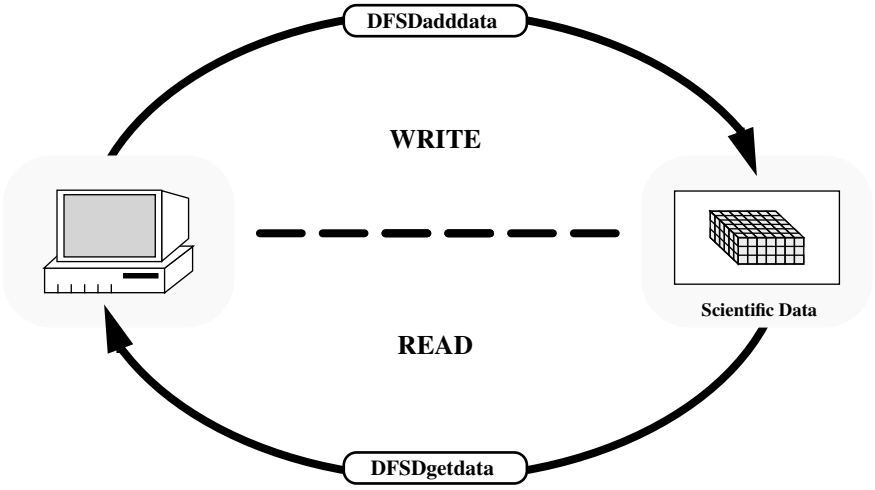
```
      PROGRAM RASTER24

      integer         d2gimg, ret, width, height, interlace
      character*1     raster_data(5,6,3)


C Read raster-24 image dimensions
      ret = d2gdims('example.hdf', width, height, interlace)

C Read raster-24 image data from file, if dimensions are correct
      if ( (width .eq. 5) .and. (height .eq. 6) ) then
          ret = d2gimg('example.hdf', raster_data, width, height)
      endif

      stop
      end
```

3.5 HDF and the Scientific Data Interface

Every function in the scientific data interface begins with the prefix ***DFSD*** which stands for "data format, scientific data." By using DFSD function calls, HDF can often make a series of assumptions about the operations you are performing. These assumptions reduce the amount of work you must do to perform a successful read or write operation.

When you make a single DFSD call to read or write scientific data, HDF assumes you are interested in reading or writing an array of 32-bit floating point data (See FIGURE 3.5a). To write the data to a file, you can use **DFSDadddata**. To read the data from a file, you must use **DFSDgetdata**.

FIGURE 3.5a **Float32 scientific data can be read or written to a file via a single read or write function call.**

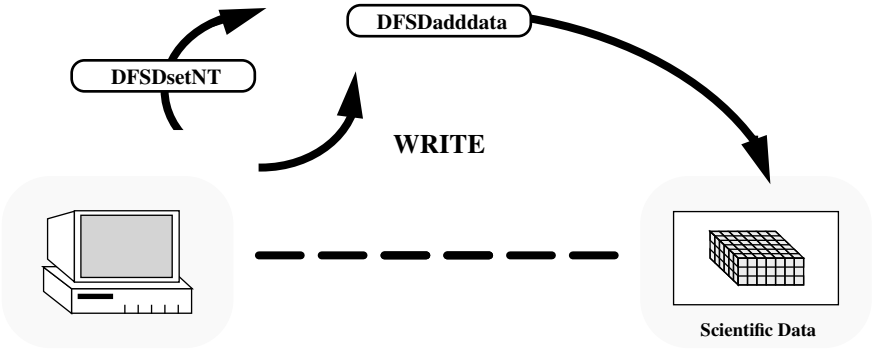


Although **DFSDadddata** and **DFSDgetdata** are the only two calls required for reading or writing 32-bit floating point data, the DFSD interface provides several additional functions for more advanced operations. For more information on additional DFSD function calls, consult the HDF User's Guide.

Writing Scientific Data to an HDF File

If you are programming in C, **DFSDadddata** is the only function call you need to write an array of 32-bit floating point data to file. In FORTRAN, the equivalent function call is **dsadata**. If you wish to store data of a different number type, you must call **DFSDsetNT** (or **dssnt** in FORTRAN) to change the default setting (See Figure 3.5b on page 34).

FIGURE 3.5b **DFSDsetNT can be used to override the DFSDadddata float32 default setting.**



If you are writing an array of 32-bit floating point data to file, the function call made from your program should look something like this:

C: `DFSDadddata (filename, rank, dimsizes, data)`
FORTRAN: `dsadata (filename, rank, dimsizes, data)`

However, if you wish to change the default number type, your program must include the following calls:

C:
```
DFSDsetNT (number_type)
DFSDadddata (filename, rank, dimsizes, data)
```

FORTRAN:
```
dssnt (number_type)
dsadata (filename, rank, dimsizes, data)
```

The parameters for both the “set number type” and the “add data” call are more explicitly defined below (See TABLE 3H). Notice that the set number type offers several alternative number types for overriding the default setting.

TABLE 3H **The parameters required for DFSDsetNT and DFSDadddata.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DFSDsetNT (dssNT)	number_type	integer	32-bit integer	Specifies the number type: DFNT_FLOAT32 or 5 = 32-bit floats DFNT_FLOAT64 of 6 = 64-bit floats DFNT_INT8 or 20 = 8-bit ints DFNT_UINT8 or 21 = 8-bit unsign ints DFNT_INT16 or 22 = 16-bit ints DFNT_UINT16 or 23 = 16-bit unsign ints DFNT_INT32 or 24 = 32-bit ints DFNT_UINT32 or 25 = 32-bit unsign ints
DFSDadddata (dsadata)	filename	character*(*)	character pointer	Name of file storing the data.
	rank	integer	32-bit integer	Number of dimensions in data array.
	dimsizes	integer array	32-bit integer array	Size of each dimension in data array.
	data	<number type> array	<number type> pointer	Array holding the data.

EXAMPLE 7.

Writing Scientific Data to an HDF file

In the following code example, **DFSDadddata** and **dsadata** are used to write an array of 64-bit floating point numbers to a file called "example.hdf." Although the **DFSDsetNT** function call is optional, it is included here to demonstrate how to override the float32 default setting. In both the C and FORTRAN example, the scientific data is initialized using the following values:

1	2	3	4	5	
6	11	12	13	14	15
	16	21	22	23	24	25
		26	27	28	29	30

C:

```
#include "hdf.h"

#define LENGTH 3
#define HEIGHT 2
#define WIDTH 5

main()
{
    /* Create data array; store dimensions in array 'dims' */
    static float64 scien_data[LENGTH][HEIGHT][WIDTH] =
                                                  { 1., 2., 3., 4., 5.,
                                                    6., 7., 8., 9.,10.,

                                                   11.,12.,13.,14.,15.,
                                                   16.,17.,18.,19.,20.,

                                                   21.,22.,23.,24.,25.,
                                                   26.,27.,28.,29.,30. };
    int32 dims[3] = {LENGTH, WIDTH, HEIGHT};

    /* Set number type to 64-bit float */
    DFSDsetNT(DFNT_FLOAT64);

    /* Write the data to file */
    DFSDadddata("example.hdf", 3, dims, scien_data);
}
```

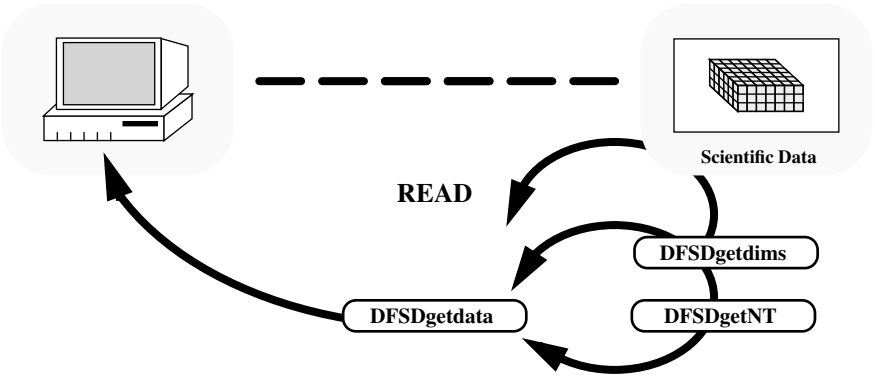
FORTRAN:

```
      PROGRAM SDS

      integer  dsadata, dssnt, dims(3), ret
      real*8   sci_data(5,2,3)

C Create data array called 'sci_data'; store dimensions in array 'dims'
      data     sci_data/             1., 2., 3., 4., 5.,
     $                               6., 7., 8., 9.,10.,
     $
     $                              11.,12.,13.,14.,15.,
     $                              16.,17.,18.,19.,20.,
     $
     $                              21.,22.,23.,24.,25.,
     $                              26.,27.,28.,29.,30./

      data dims /5,2,3/

C Set number type to 64-bit float
      ret = dssnt(6)

C Write the data to file
      ret = dsadata('example.hdf', 3, dims, sci_data)

      stop
      end
```

Reading Scientific Data from an HDF File

If you are programming in C, **DFSDgetdata** is the only function call you need to make to read a 32-bit floating point array from a file**.** In FORTRAN, the equivalent function is called **dsgdata**. If you are not sure of the dimensions or the number type for the array you intend to read, **DFSDgetdims** and **DFSDgetNT** (or **dsgnt** and **dsgdims** in FORTRAN) can obtain them for you. Both calls are optional, however **DFSDgetNT** will only work if **DFSDgetdims** is called first. By using both **DFSDgetdims** and **DFSDgetNT** in your program, you can verify both the size of the space required to hold the data as well as its number type (See FIGURE 3.5c).

FIGURE 3.5c **DF24getdims and DF24getNT will return the dimensions and number type of an array.**



If you know the dimensions and number type for the array you intend to read, the function call made from your program should look something like this:

C:
```
DFSDgetdata (filename, rank, dimsizes, maxrank)
```
FORTRAN:
```
dsgdata (filename, rank, dimsizes, maxrank)
```

However, if you wish to verify the dimensions and number type of an array, your program should include the following calls:

C:
```
DFSDgetdims (filename, rank, dimsizes, maxrank)
DFSDgetNT (number_type)
DFSDgetdata (filename, rank, dimsizes, data)
```

FORTRAN:
```
dsgNT (filename, rank, dimsizes, maxrank)
dsgdims (number_type)
dsgdata (filename, rank, dimsizes, data)
```

The first time the “get dims” and “get number type” call are made, they return information about the first array of scientific data in the named file. Any subsequent calls return the same information for successive arrays of scientific data in the file. The “get data” call operates in a similar manner. The parameters for the “get dims,” “get number type,” and the “get data” call are more explicitly defined below (See TABLE 3I on page 37).

TABLE 3I **The parameters required for DFSDgetNT, DFSDgetdims, and DFSDgetdata**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
DFSDgetdims (dsgdims)	filename	character* (*)	character pointer	Name of file containing the data.
	rank	integer	integer	Number of dimensions.
	dimsizes	integer	32-bit integer	Array holding the dimension sizes.
	maxrank	integer	integer	Size of array containing the dim sizes.
DFSDgetNT (dsgnt)	number_type	integer	32-bit integer	The number type for the data.
DFSDgetdata (dsgdata)	filename	character* (*)	character pointer	Name of file containing the data.
	rank	integer	32-bit integer	Number of dimensions.
	maxsizes	integer*4 array	32-bit integer array	Array holding the dimension sizes.
	data	<number type> array	<number type> pointer	Array to hold the scientific data.

EXAMPLE 8. **Reading an array of scientific data from an HDF file**

The following example queries a file called “example.hdf” for the dimensions and number type of an array containing scientific data. Although **DFSDgetdims** and **DFSDgetNT** are optional, they are included here as a demonstration of how to check the array dimensions and number type before reading any data. If the dimension and type are known, only the **DFSDgetdata** call is required.

C:

```
#include “hdf.h”

#define LENGTH 3
#define HEIGHT 2
#define WIDTH 5

main()
{
    float64 scien_data[LENGTH][HEIGHT][WIDTH];
    int32 number_type;
    intn rank;
    int32 dims[10];

    /* Get the dimensions and number type of the array */
    DFSDgetdims("example.hdf", &rank, dims, 10);
    DFSDgetNT(&number_type);

    /* Read the array if the dimensions are correct */
    if(dims[0] <= LENGTH && dims[1] <= HEIGHT && dims[2] <= WIDTH)
        DFSDgetdata("example.hdf", rank,dims, scien_data);
 }
```

FORTRAN:

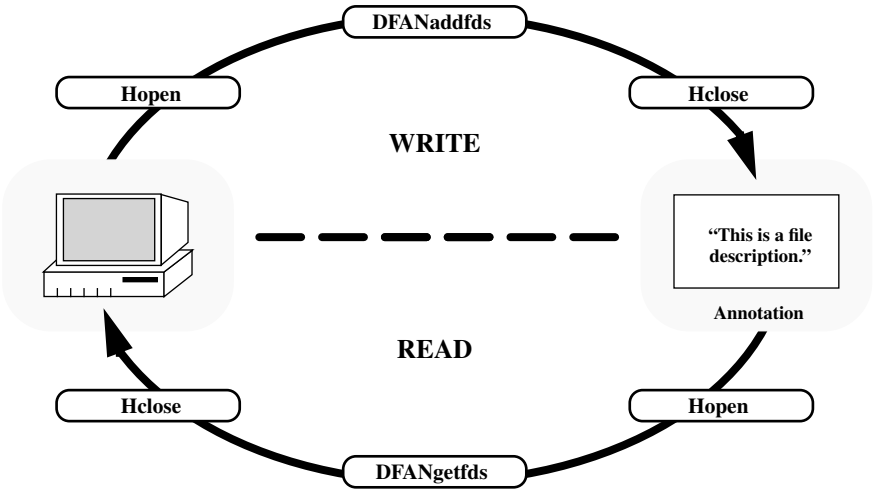
```
      PROGRAM SDS

      integer  dsgdata, dim(3), ret
      real*8   sci_data(5, 2, 3)

C Get the dimensions and number type of the array
      ret = dsgdims('example.hdf', rank, dim, 3)
      ret = dsgnt(num_type)

C Read the array if the dimensions are correct
      if((dim(1) .eq. 3) .and. (dim(2) .eq. 2) .and. (dim(3) .eq. 5)) then
          ret = dsgdata('example.hdf', 3, dims, sci_data)
      endif

      stop
      end
```

3.6 HDF and the Annotation Interface

Every function in the annotation interface begins with the prefix ***DFAN*** which stands for "data format, annotation." By using DFAN function calls, you can annotate a file with a file label (abbreviated as **fid**) or a file description (abbreviated as **fds**). Here the difference between labels and descriptions is that labels are terminated by zero or NULL, whereas descriptions are not.

Even though a reading or writing a file annotation requires only one call to the DFAN interface, both operations require two calls to the general HDF interface (See FIGURE 3.6a). Annotation operations rely on **Hopen** and **Hclose** function calls to open and close the appropriate HDF files. To begin an annotation operation, **Hopen** must open the named file and pass its corresponding file_id to the DFAN interface. Once the file is open and an annotation is written via a **DFANaddfds** call or read via a **DFANgetfds** call, **Hclose** completes the operation by closing the newly annotated file.

FIGURE 3.6a **Any HDF file can be annotated via one read or write DFAN function call.**



Although **DFANaddfds** and **DFANgetfds** are the only two DFAN calls required to read or write a file description, the DFAN interface provides several other functions for more advanced operations. For more information on annotating data elements and other useful annotation operations, consult the HDF User's Guide.

Writing a File Description

If you are programming in C, **DFANaddfds** is the only DFAN function call you will need to create a file description**.** In FORTRAN, the equivalent function is called **daafds**. Hopen must precede your "add file description" call, and Hclose must follow it. To add an annotation to a file, your program should contain a series of function calls something like this:

C:
```
Hopen (filename, access, block_size)
DFANaddfds (file_id, text, text_length)
Hclose (file_id)
```

FORTRAN:
```
hopen (filename, access, block_size)
daafds (filename, text, text_length)
hclose (file_id)
```

The parameters for the these functions are defined below (See TABLE 3J). To include additional annotations in a file, you must assign each additional annotation to an individual element. To learn more about annotating data elements, see the HDF User's Guide.

TABLE 3J **The parameters for the DFAN function calls required to write a file description.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
Hopen	filename	character*(*)	character pointer	Name of the file to be accessed.
	access	integer	integer	Operation to be performed: DFACC_READ or 1 = read only DFACC_RDWR or 3 = read/write DFACC_CREATE or 4 = create/overwrite
	block_size	integer	16-bit integer	0: if adding data to an existing file.
DFANaddfds (daafds)	file_id	integer	32-bit integer	The file identification for the opened file.
	text	character*(*)	character pointer	Text of the description.
	text_length	integer	32-bit integer	Length of the description in bytes.
Hclose	file_id	integer	32-bit integer	The file identification for the opened file.

EXAMPLE 9. **Writing a File Description to an HDF file**

The following example adds a text description to an HDF file called "example.hdf."

C:

```
#include "hdf.h"

main()
{
    int32 file_id;
    static char file_descript[] = "This is a file description";

    /* Open the HDF file to write an annotation */
    file_id = Hopen("example.hdf", DFACC_RDWR, 0);

    /* Use the file_id to write the description to a file */
    DFANaddfds(file_id, file_descript, 40);

    /* Close the file */
    Hclose(file_id);

}
```

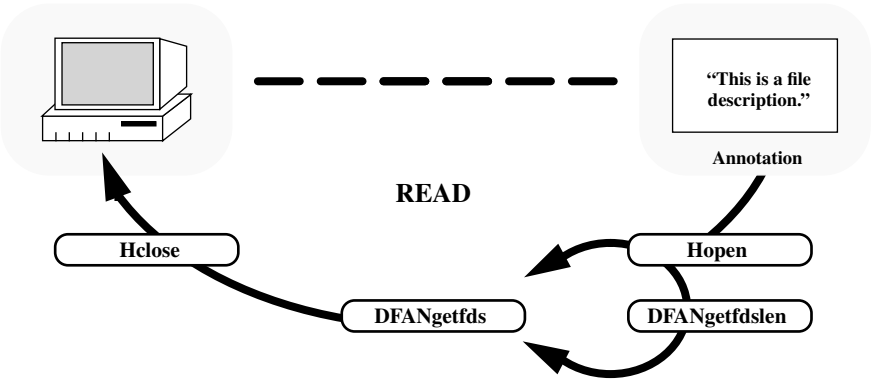
FORTRAN:

```
      PROGRAM ANNOTATION

      integer        daafds, ret, file_id, hopen, hclose, DFACC_RDWR
      character*256  f_descr

      parameter (DFACC_RDWR = 3)
      f_descr = 'This is a file description'

C Open the HDF file to write an annotation
      file_id = hopen('example.hdf', DFACC_RDWR, 0)

C Use the file_id to write the description to a file
      ret = daafds(file_id, f_descr, 40)

C Close the file
      ret = hclose(file_id)

      stop
      end
```

Reading a File Description

If you are programming in C, **DFANgetfds** is the only DFAN function call you need to read a file description**.** In FORTRAN, the equivalent function is called **dagfds**. If you are not sure of the length of the description you are attempting to read, the **DFANgetfdslen** function (or **dagfdsl** in FORTRAN) can obtain it for you. **DFANgetfdslen** is optional. However, by using **DFANgetfdslen** in your program, you can verify the size of the space required to hold the description before reading any data. In either case, Hopen must precede the "add file description" call, and Hclose must follow it (See FIGURE 3.6b).

FIGURE 3.6b **DFANgetfdslen can be used before DFANgetfds to obtain the length of an annotation.**



If you already know the length of the annotation you intend to read, the series of function calls made from your program should look something like this:

C:
```
Hopen (filename, access, block_size)
DFANgetfds (filename, description, desc_length, isfirst)
Hclose (file_id)
```

FORTRAN:
```
hopen (filename, access, block_size)
dagfds (filename, description, desc_length, isfirst)
hclose (file_id)
```

However, if you wish to verify the string length before reading the annotation, your program should include the following:

C:
```
Hopen (filename, access, block_size)
DFANgetfdslen (filename, isfirst)
DFANgetfds (filename, description, desc_length, isfirst)
Hclose (file_id)
```

FORTRAN:
```
hopen (filename, access, block_size)
dagfdsl (filename, isfirst)
dagfds (filename, description, desc_length, isfirst)
hclose (file_id)
```

The parameters for these function calls are more explicitly described in the table below (See TABLE 3K on page 41).

TABLE 3K **The parameters required for DFANgetfdslen and DFANgetfds.**

Function Call	Parameter	Variable Type		Description
		FORTRAN	C	
Hopen	filename	character*(*)	character pointer	Name of the file to be accessed.
	access	integer	integer	Operation to be performed: DFACC_READ or 1 = read only DFACC_RDWR or 3 = read/write DFACC_CREATE or 4 = create/overwrite
	block_size	integer	16-bit integer	0: if adding data to an existing file.
DFANgetfdslen (dagfdsl)	file_id	integer	32-bit integer	The file identification for the opened file.
	isfirst	integer	integer	1: start with the first annotation. 0: get the length of the next annotation.
DFANgetfds (dagfds)	file_id	integer	32-bit integer	The file identification for the opened file.
	description	character array	character array	Text of the description.
	desc_length	integer	32-bit integer	Length of the description.
	height	integer	integer	1: start with the first annotation. 0: get the length of the next annotation.
Hclose	file_id	integer	32-bit integer	The file identification for the opened file.

EXAMPLE 10.

Reading a File Description

The following example reads a file description from an HDF file called “example.hdf.” Although the **DFANgetfdslen** call is optional, it is included in the following example as a demonstration of how to check the length of a description before reading its data. If the length is already known, only the **DFANgetfds** call is required.

C:

```
#include “hdf.h”

main()
{
    int32 file_id, file_descript_len;
    char file_descript[100];

    /* Open the HDF file containing the annotation */
    file_id = Hopen("example.hdf", DFACC_READ, 0);

    /* Determine the length of the file description */
    file_descript_len=DFANfdslen(file_id,1);

    /* Read the file annotation if the length is correct */
    if(file_descrip_len<=100)
        DFANgetfds(file_id,file_descript,file_descript_len,1);

    /* Close the file */
    Hclose(file_id);
}
```

FORTRAN:

```
      PROGRAM ANNOTATE

      integer        dagfdsl, dagfds, ret, file_id, f_ds_len
      integer        hopen, hclose, DFACC_READ
      character*100  f_descr

      parameter (DFACC_READ = 1)

C Open the HDF file containing the annotation
      file_id = hopen('example.hdf', DFACC_READ, 0)

C Determine the length of the file description
      f_ds_len = dagfdsl(file_id, 1)

C Read the file annotation if the length is correct
      if ((f_ds_len .le. 100)) then
          ret = dagfds(file_id, f_descr, 100, 1)
      endif

C Close the file
      ret = hclose(fileid)

      stop
      end
```