The two functions shown in Figure 2.15 convert a floating-point data array into an 8-bit raster array. Once converted, this raw raster array is ready to be stored in RIS8 format.

Figure 2.15 Converting Floating-Point Data to RIS8

C:

```
/* floattor8.c
 */
#include "df.h"

#define CHAR_MAX      255

/*
 * floattoR8
 * Convert a data array into a raster array by dividing the
 * range in the data into 256 regions, numbering  the regions
 * from zero to 255, and assigning to each position in the
 * raster array the number of the corresponding region.
 *
 * Assume that raster array allocated is big enough.
 */
floattoR8(data, raSter, size, max, min)
float data[];
char  raster[];
int   size;
float max, min;
{
  int32 i;
  float32 step;

  if ((max == 0) && (min == 0))
    findMaxMin(data, size, &max, &min);

  step = (max - min) / CHAR_MAX;
  if (step == 0)
    return(-1);

  for(i=0;i<size;i++)
    raster[i] = (char) ((data[i] - min) / step);
}

/*
 * findMaxMin
 * Finds the maximum and minimum values in a data array.
 */
findMaxMin(data, size, max, min)
float32 data[];
int32 size;
float32 *max, *min;
{
  int32 i;

  *max = *min = data[0];
  for(i=1;i<size;i++) {
    if (*max < data[i])
      *max = data[i];
    else if (*min > data[i])
      *min = data[i];
  }
}
```

- Writes the palette and image as a run-length encoded raster image set to a file called `testrig1.hdf`
- Reads the palette and image back in, this time storing the image in `image2` and storing the palette back in `palette`
- Compares the contents of `image2` to the contents of `image1` to determine whether they are identical, as they should be

Figure 2.14 C Program Dealing with Raster Image Sets

```
C:
#include "df.h"

main
{
               char
               image1[131072],              /* raw image to be read in from denaa031,
                                               then put into an RIS8 in testrig.df */
                      image2[131072],       /* image to be read in from testrig.df */
                      palette[768],
                      reds[256],            /* colors to be loaded into palette */
                      greens[256],
                      blues[256],
               *p;                          /* pointer to palette */
               int j;
               int width, height,
                      ispal;                /* boolean to tell if there is a palette    */
               FILE *fp;

               fp = fopen("denaa031","r");  /* read in raw 256x512 image */
                      fread(image1, 131072, 1, fp);
                      fclose(fp);
               fp = fopen("ps.pal","r");    /* read RGB values from palette file */
                      fread(reds,1,256,fp);
                      fread(greens,1,256,fp);
                      fread(blues,1,256,fp);
                      fclose(fp);

               p = palette;
               for (j=0; j<256; j++) {      /* reorganize palette so that */
                      *p++ = reds[j];       /* RGB values are interleaved */
                      *p++ = greens[j];
                      *p++ = blues[j];
               }

               printf("Ready to write out image \n");
               DFR8setpalette(palette);
               DFR8putimage("testrig1.df",image1,256,512,DFTAG_RLE);
               printf("Just wrote out image \n");

               DFR8getdims("testrig1.df",&width, &height, &ispal);
               printf("After getdim\n");
               printf("\tdimensions:%d :%d\n\tispal: %d\n", width,height,ispal);
               DFR8getimage("testrig1.df",image2,width,height,palette);
               printf("After getimage ... ");
               if (memcmp(image1, image2, 131072) ==0) printf("identical\n");
               else printf("different\n");
}
```

C Functions to Convert Floating-Point Data to 8-Bit Raster Data

Sample Programs

The following sections contain various FORTRAN and C programs that use HDF functions.

A FORTRAN Program to Copy a RIS8 from One File to Another

This program reads into `image1` an 8-bit raster image from an HDF file called '`testrig1.df`'. At the same time, it reads a palette into the array `palette`. The dimensions of the image are read into `width` and `height`. It is assumed that the image is small enough to fit into the 150,000-character array `image1`; i.e., the value `width*height` must be less than 150,000.

The program next writes out the palette and image to the file '`testrig2.df`'. Since the `compress` parameter in `DFR8putimage` is 12, the output image is encoded using run length encoding.
(Figure 2.13)

Figure 2.13 FORTRAN Program to Copy a RIS8 from One File to Another

FORTRAN:
```
PROGRAM RISexample

CHARACTER*1    image1(150000)
CHARACTER*1    palette(768)
INTEGER        width, height, ispal
INTEGER        ret


ret = DFR8getdims('testrig1.df', width, height, ispal)
ret = DFR8getimage('testrig1.df',image1,width,height,palette)

ret = DFR8setpalette(palette);
ret = DFR8putimage('testrig2.df',image1,width,height,12)

stop
end
```

A C Program to Convert a Raw Palette and Raw Raster Image to HDF RIS8 Format

The example in Figure 2.14 shows a complete program for processing RIS8 data. Several features of HDF image storage are illustrated here. The program does the following, in order:

- Reads into `image1` a 256 x 512 8-bit raster image from a non-HDF file called `denaa031`

- Reads into each of `red`, `green`, and `blue` 256 values representing the palette from a file called `ps.pal` (The palette stored in `ps.pal` is not in HDF format, so it is rearranged into proper format in a new, 768-byte array called `palette`.)

Figure 2.11 Reading an RIS24: Dimensions and Interlace Known

```
C:
int            DF24getimage;
char           image[3][256][512];

DF24getimage("myfile.hdf",image,256,512);
   .
   .
   .
```

Remarks:

- The RIS24, stored in a file called '`myfile.hdf`', is read into an array called `image`.
- The raster image stored in the file is known to be 256 x 512.
- The array `image` is a 3 x 256 x 512 array of 8-bit characters.

Example: Read an Image, Dimensions and Interlace Not Known

Figure 2.12 illustrates a set of C calls that read in an image, where the dimensions of the image and interlace scheme are not known ahead of time.

Figure 2.12 Reading an RIS24: Dimensions and Interlace Not Known

```
C:
int            DF24getdims, DF24getimage;
int            width, height, il;
char           *image;  /* pointer to space to return image */

DF24getdims("myfile.hdf",&width,&height,&il);
DF24reqil(2);
image = (char *) malloc(3*width*height);
DF24getimage("myfile.hdf",image,width,height);
   .
   .
   .
```

Remarks:

- The RIS24 , stored in a file called '`myfile.hdf`', is read into an array called `image`.
- The data is stored in the array `image` as if the array were three planes of size *`width`* x *`height`*.
- Since no explicit declaration is given for `image`, it is the responsibility of the program to compute offsets in the array that correspond to particular elements.

DFANgetlablist, which returns a list of labels for a given tag together with their reference numbers. It provides, in a sense, a random access to images.

NOTE: There is no guarantee that reference numbers appear in sequence in an HDF file; therefore, it is not safe to assume that a reference number is the sequence number for an image.

DF24reqil

FORTRAN:
```
INTEGER FUNCTION DF24reqil(il)
INTEGER il               - interlace to get next image with
```

C:
```
int DF24reqil(il)
int il;                /* interlace to get next image with */
```

Purpose: To cause next DF24getimage to store image in memory with the specified interlace.

Returns: 0 on success; –1 on failure with DFerror set.

Regardless of what interlace scheme is used to store the image, DF24reqil causes the image to be loaded into memory and be interlaced according to the specification of il.

NOTE: Since a call to DF24reqil may require a substantial reordering of the data, I/O performance could be adversely affected; e.g. it could result in much slower I/O performance than would be achieved if no change in interlace were requested.

Interlace codes: 0 = pixel interlacing; 1 = scan-line interlacing;
2 = scan-plane interlacing.

DFR8restart

FORTRAN:
```
INTEGER FUNCTION DFR24restart()
```

C:
```
int DFR24restart()
```

Purpose: To cause the next get to read from the first RIS24 in the file, rather than the RIS24 following the one that was most recently read.

Returns: 0 on success; -1 on failure

Example: Reading in a 24-bit Image

Figure 2.11 shows a C call that reads in an image when the dimensions and interlace are already known.

Interlace codes: 0 = pixel interlacing; 1 = scan-line interlacing; 2 = scan-plane interlacing.

DF24getimage

FORTRAN:

```
INTEGER FUNCTION DF24getimage(name, image, width, height)
CHARACTER*(*) name        -  name of HDF file
CHARACTER*(*) image       -  pointer to space to return image
INTEGER width, height     -  dimensions of space to redurn image
```

C:

```
int DF24getimage(filename, image, width, height)
char *filename;       /* name of HDF file */
char *image;          /* pointer to space to return image */
int32 width, height;  /* dimensions of space to return image */
```

Purpose: To get image from next 24-bit RIS.

Returns: 0 on success; –1 on failure with `DFerror` set.

If `DFR24getdims` has not been called, `DFR24getimage` finds the next image in the same way that `DFR24getdims` does.

The amount of space allocated for the image should be *width* x *height* x 3 bytes.

To specify that the next call to `DF24getimage` should read the raster image from the RIS24 using a particular interlace, rather than the interlace used to store the image in the file, make a call to `DF24reqil` (see below).

DF24readref

FORTRAN:

```
INTEGER FUNCTION
DF24readref(name, ref)
character*(*) name     - name of file containing image
integer       ref      - reference number for next DF24getimage
```

C:

```
int
DF24readref(filename, ref)

char *filename;     /* file containing image */
uint16 ref;               /* reference number for next DF24getimage
*/
```

Purpose: To specify the reference number of the image to be read when `DF24getimage` is next called.

Returns: 0 on success; -1 on failure.

You will most likely use this routine in conjunction with

Figure 2.10 Storing Multiple RIS24s in a Single File

```
C:
int           DF24addimage
char          pic1[800][1200][3], pic2[800][1200][3]
char          pic3[800][3][1200], pic4[800][3][1200]

DF24addimage('myfile',pic1,800,1200)
DF24addimage('myfile',pic2,800,1200)
DF24setil(1)
DF24addimage('myfile',pic3,800,1200)
DF24addimage('myfile',pic4,800,1200)
  .
  .
  .
```

Reading 24-Bit Raster Images from a File

The two routines, `DF24getdims` and `DF24getimage`, are sufficient to read raster images from a file. If enough is known about the images and interlacing, only the latter routine is needed.

DF24getdims

```
FORTRAN:
INTEGER FUNCTION DF24getdims(name, width, height, il)
CHARACTER*(*) name   -  name of HDF file
INTEGER width, height   - for returning dimensions
INTEGER il           -  for returning inteRlace of image in file
```

```
C:
int DF24getdims(filename, pwidth, pheight, pil)
char *filename;           /* name of HDF file */
int32 *pwidth, *pheight;  /* for returning dimensions */
int *pil;                 /* for returning interlace of image
                             in file */
```

Purpose: To get dimensions and interlace storage scheme of next image RIS.

Returns: 0 on success; –1 on failure with `DFerror` set.

If the file is being opened for the first time, `DF24getdims` returns information about the first image in the file. If an image has already been read, `DF24getdims` finds the next image. Thus, images are read in the same order in which they were written to the file.

If you know the dimensions of the image beforehand, there is no need to call `DF24getdims`. Simply allocate arrays with the proper dimensions for the image and let `DF24getimage` read in the images. If, however, you do not know the values of `width` and `height`, you must call `DF24getdims` to get them and then use them to determine the right amount of space needed for the array `image`.

Successive additional calls to `DF24getdims` and `DF24getimage`, respectively, retrieve all of the images in the file in the sequence in which they were written.

DF24addimage

```
FORTRAN:
INTEGER FUNCTION DF24addimage(name, image, width, height)
CHARACTER*(*) name        - name of HDF file
CHARACTER*(*) image       - array for return image
INTEGER width, height     - dimensions of array image
```

```
C:
int DF24addimage(filename, image, width, height)
char *filename;         /* name of HDF file */
char *image;            /* pointer to array for return image */
int32 width, height;    /* dimensions of array image */
```

Purpose: To write out a 24-bit image.

Returns: 0 on success; –1 on failure with `DFerror` set.

Array `image` is assumed to be *`width`* x *`height`* x 3 bytes.

Example: Writing a RIS24 with the Default Pixel Interlace

The C code in Figure 2.9 demonstrates how the default pixel interlace is used when storing the 400 x 600 arrAy `picture` in a file in RIS24 format.

Figure 2.9 Storing an RIS24 Using Pixel Interlace

```
C:
int             DF24addimage;
char            picture[3][600][400];
int             ret;

ret = DF24addimage("herfile.hdf",picture,600,400);

if (ret != 0)
    printf("Error writing image to myfile.hdf.");
  .
  .
  .
```

Example: Writing Several 24-bit Images

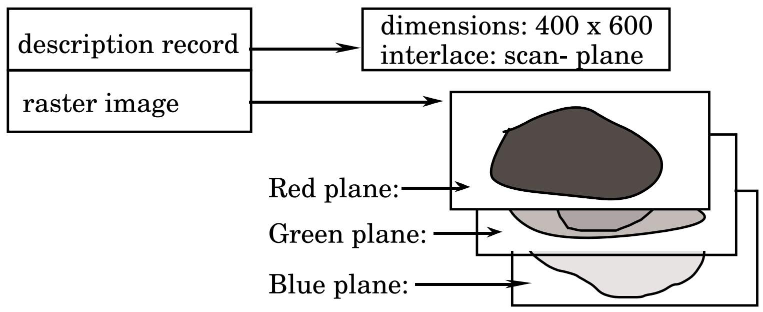
Figure 2.10 shows a series of C calls in which four 800 x 1200 images are written to the same file. The first two calls use the default interlace scheme; the second two calls use scan-line interlace.

three components (R, G, and B) informs NCSA HDF to assume that the image is stored as an array of size 100 x 200 x 3.

Specifically, an interlace code of 2 indicates that the bytes that describe the image are stored in the following order in the file or memory: R values are stored in the first 100 x 200 bytes of the array for each of the pixels in the image; the G values for the image are stored in the second 100 x 200 plane; and the B values in the third.

Figure 2.8 illustrates how an RIS24 — stored using the scan-plane interlace — looks in an HDF file.

Figure 2.8 Scan-Plane Interlace



Compression Schemes

As of this writing, image compression has not been implemented for 24-bit images in HDF. However, there are plans to implement a routine to cause 24-bit images to be stored in compressed mode. This routine should be available in the next release of HDF.

Writing 24-Bit Raster Images to a File

DF24setil

FORTRAN:

```
INTEGER FUNCTION DF24setil(il)

INTEGER il;              – interlace of image
```

C:

```
int DF24setil(il)

int il;               /* interlace of image */
```

Purpose: To set interlace scheme to be used on subsequent writes.

Returns: 0 on success; –1 on failure with `DFerror` set.

If `DF24setil` is not called, the interlace code is assumed to be 0. Interlace codes: 0 = pixel interlacing; 1 = scan-line interlacing;
2 = scan-plane interlacing.

Table 2.3 Interlace Scheme Codes

Value of `il`	Interlace Scheme
0	pixel
1	scan-line
2	scan-plane

Pixel Interlace Scheme

The default interlace scheme describes an image pixel-by-pixel. This scheme is called *pixel interlace*. The code to specify the pixel interlace scheme in an RIS24 is 0 (zero).

For example, by default, NCSA HDF assumes that a 100 x 200 image with three components (R, G, and B) is stored as an array of size 3 x 100 x 200, and that each element of this array is exactly one byte in size and contains an R, G, or B value.

Specifically, an interlace code of 0 indicates that the bytes that describe the image are stored in the following order in the file or memory:

R, G, and B values are stored, in that order, in the first three bytes of the first row of the array, corresponding to the first pixel in the first row of the image.

R, G, and B values are stored in the second three bytes of the first row of the array, corresponding to the second pixel in the first row of the image.

And so forth, until the RGB values for the 100 pixels of the first row of the image are stored. This process is repeated until the RGB values for each pixel in the 200 lines of the image are stored.

Scan-Line Interlace Scheme

The *scan-line interlace scheme* describes an image line-by-line. The code to specify the scan-line interlace scheme is 1.

For example, an interlace scheme code of 1 for a 100 x 200 image with three components (R, G, and B) informs NCSA HDF to assume that the image is stored as an array of size 100 x 3 x 200.

Specifically, an interlace code of 1 indicates that the bytes that describe the image are stored in the following order in the file or memory: 100 R values are stored consecutively in the first row of the array for each of the pixels in the first line of the image, then 100 G values are stored in the second row of the array for each of the pixels in the first line of the image, then 100 B values are stored in the third row of the array for each of the pixels in the first line of the image, and so forth, until the RGB values for each of the 200 lines of pixels in the image are stored.

Scan-Plane Interlace Scheme

The *scan-plane interlace scheme* describes an image color component-by-color component. The code to specify the scan-plane interlace scheme is 2.

For example, an interlace scheme code of 2 for a 100 x 200 image with

The RIS8 interface only supports writing an image from an array that was allocated to be exactly the same size of the image.

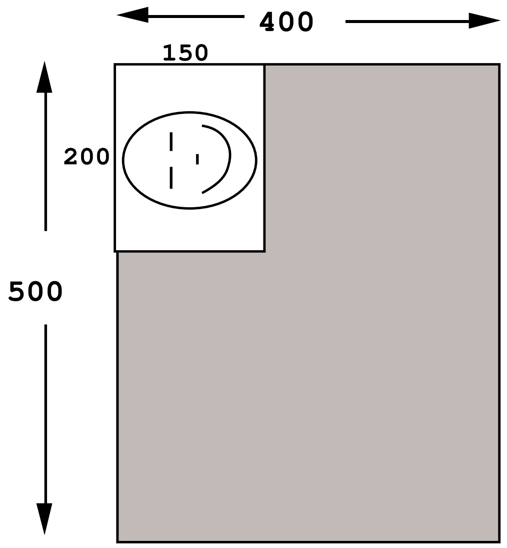


Figure 2.7 FORTRAN Image Stored in Oversized Buffer

24-Bit Raster Image Sets

The phrase *24-bit raster image set* (RIS24) refers to the set of tags and associated information required to store a 24-bit raster image in an HDF file. An RIS24 contains at least the following components:

- An *image*—here, a two-dimensional array of 24-bit pixel representations, where each 24-bit pixel value has three 8-bit components: one each for the red, green, and blue (RGB) values of the pixel color. These RGB values may be arranged in the file in one of three different ways (see the following section, "Interlace Schemes").
- An *interlace scheme*—a code that describes the order in which the pixel components are physically stored in the file (see the following section, "Interlace Schemes").
- *Dimensions*—two values that represent the x and y dimensions of the image, respectively.

Interlace Schemes

An *interlace scheme* describes the way an image is stored in a file or in memory. NCSA HDF supports different interlace schemes because graphics applications and devices vary in the way they organize graphics images. By storing an image in a file using a scheme that is consistent with the expected application or device, you can achieve substantial improvements in performance. The value of the integer argument `il` determines which scheme is to be used, as shown in Table 2.3. The interlace schemes are described in the following sections.

150 bytes high. It is assumed that the array `image` is dimensioned to be exactly the correct size. Alternatively, after `DFR8getdims` has read in the height and gidth, space for it could be allocated by the program.

- The parameter `pal` is a 768-byte array, three bytes per color, representing R, G, and B values, respectively, for each color.

Example: Reading in a Raster Image Set

Figure 2.6 shows a FORTRAN program that reads in an image whose size is different from the size of the space allocated to hold the image. It then moves the image to a array that is of the correct size, and outputs the new image to a new HDF file.

Figure 2.6 Reading an RIS8: Dimensions Different from Allocated Space

FORTRAN:

```
      program big_buffer

C Program using DFR8getimage when buffers are larger than image
C

C****||**************************************************
      integer DFR8getdims, DFR8getimage
      integer width, height, ispal, ret
      character*1 image(500,400)
      character*1 newimage(200,150), pal(768)

C****||****** read in image into "too-large" array ********
      ret = DFR8getdims('old.hdf', width, height, ispal)
      ret = DFR8getimage('old.hdf', image,500, 400, pal)
      print *, 'width=',width,'   height=',height

C****||*** copy image to an array that is the "right size" ***
      do 100 i=1,200
          do 100 j=1,150
              newimage(i,j) = image(i,j)
  100 continue

C****||*** write newimage new file—same as original image ***
      ret = DFR8putimage('new.hdf',newimage,200,150,0)

      stop
      end
```

Remarks:

- The RIS8, stored in a file called, `old.hdf`, is read into an array called `image`. This array is deliberately made larger than the expected image. Figure 2.7 shows how the image fits inside the buffer. Note that since FORTRAN is used, the image is stored "on its side."

- The array `newimage` is an array that is exactly the size of the image. After it is written to `new.hdf`, you can view the image in that file and see that it is identical to the original image.

- Although it is possible, as this example illustrates, to read an image into a buffer that is larger than the image, the reverse is not possible.

- The raster image stored in the file is known to be 200 bytes wide and 150 bytes high. Because of the storage order used by FORTRAN, the program is loading the image "on its side." Hence, the declaration "`CHARACTER*1 image(200,150)`" rather than "`CHARACTER*1 image(150,200).`"

- It is assumed that the array `image` is dimensioned to be exactly the correct size.

- The parameter `pal` is a 768-byte array, three bytes per color, representing R, G, and B (red, green, and blue) values, respectively, for each color.

- If `DFR8getimage` executes successfully, the return value 0 is assigned to `ret`; otherwise, –1 is assigned. In this example, the value of `ret` is not used.

Example: Reading in a Raster Image Set (C)

Figure 2.5 shows a C program that reads in an image when the dimensions are already known, and a palette is known to exist. The program also writes out the image to a new file.

Figure 2.5 Reading an RIS8: Dimensions and Presence of Palette Known (C)

```
C:
/*
** Program to illustrate use of DFR8getdims and DFR8getimage
*/
#define HEIGHT 150
#define WIDTH  200

main()
{
    int DFR8getdims(), DFR8getimage(), DFR8putimage();
    int ispal, ret, width, height;
    char image[HEIGHT][WIDTH], pal[768];

/************* read in image ****************/
    DFR8getdims("old.hdf", &width, &height, &ispal);
    if ( (width==WIDTH) && (height==HEIGHT) ) {
       DFR8getimage("old.hdf",image,width,height,pal);
    } else {
       printf("Wrong dimensions.  Program aborted.");
       exit(1);
    }
/****** write same image to different file ***********/
    DFR8addimage("new.hdf", image, width, height, 0);

}
```

Remarks:

- The RIS8, stored in a file called, '`old.hdf`', is read into an array called `image`.

- The raster image stored in the file is known to be 200 bytes wide and

DFR8lastref

FORTRAN:

```
(not yet implemented in FORTRAN)
```

C:

```
int DFR8lastref()
```

Purpose: To get last reference number written or read for an RIS8.

Returns: Reference number on success; -1 on failure.

This routine is primarily used for annotations. See Chapter 5, "Annotating Data Objects and Files," for examples.

Example: Reading in a Raster Image Set (FORTRAN)

Figure 2.4 shows a FORTRAN program that reads in an image when the dimensions are already known, and a palette is known to exist. The program also writes out the image to a new file.

Figure 2.4 Reading an RIS8: Dimensions and Presence of Palette Known (FORTRAN)

FORTRAN:

```
      program test_getimage

C Program to illustrate use of DFR8getdims and DFR8getimage
C

C****||***************************************************
      INTEGER DFR8getdims, DFR8getimage, DFR8addimage
      INTEGER ispal, ret, width, height
      CHARACTER*1 image(200,150), pal(768)

C****||***************** read in image ********************
      ret = DFR8getdims('old.hdf', width, height, ispal)
      if ( (width.eq.200) .and. (height.eq.150) ) then
         ret = DFR8getimage('old.hdf',image,width,height,pal)
      else
         print *, 'Wrong dimensions.  Program aborted.'
         stop
      endif

C****||***** write same image to different file ************
      ret = DFR8addimage('new.hdf',image, width,height, 0)

      stop
      end  .
```

Remarks:

- The RIS8, stored in a file called, '`old.hdf`', is read into an array called `image`.

DFR8readref

FORTRAN:

```
INTEGER FUNCTION
DFR8readref(name, ref)
character*(*) name     - name of file containing image
integer       ref      - reference number for next DFR8getimage
```

C:

```
int
DFR8readref(filename, ref)

char *filename;   /* file containing image */
uint16 ref;       /* reference number for next DFR8getimage */
```

Purpose: To specify the reference number of the image to be read when `DFR8getimage` is next called.

Returns: 0 on success; -1 on failure.

This routine is most likely to be used in conjunction with `DFANgetlablist`, which returns a list of labels for a given tag together with their reference numbers. It provides, in a sense, a random access to images.

NOTE: There is no guarantee that reference numbers appear in sequence in an HDF file; therefore, it is not safe to assume that a reference number is the sequence number for an image.

DFR8restart

FORTRAN:

```
INTEGER FUNCTION DFR8restart()
```

C:

```
int DFR8restart()
```

Purpose: To cause the next `get` to read from the first RIS8 in the file, rather than the RIS8 following the one that was most recently read.

Returns: 0 on success; -1 on failure

DFR8nimages

FORTRAN:

```
(No FORTRAN version is currently available.)
```

C:

```
int DFR8nimages(filename)

char *filename;       /* name of HDF file */
```

Purpose: To count the number of images contained in an HDF file.

Returns: Number of images on success; -1 on failure.

DFR8getimage

FORTRAN:

```
INTEGER FUNCTION
DFR8getimage(filename, image, bufwidth, bufheight, palette)

CHARACTER*(*)  filename         –  name of file with RIS8 image
INTEGER        bufwidth,bufheight  – dimensions of the buffer
                                    allocated to store image
CHARACTER*1 image(bufwidth,bufheight) – array that will hold
                                         image
CHARACTER*1 palette(768)        –  palette to go with image
```

C:

```
int
DFR8getimage(filename, image, bufwidth, bufheight, palette)

char     *filename;           /* name of file with RIS8
                                 image */
int32    bufwidth, bufheight;  /* dimensions of the buffer
                                 allocated to store image */
char     image[bufheight][bufwidth];       /* array that will
                                 hold image */
char     palette[768];        /* palette to go with image */
```

Purpose: To retrieve the image and its palette, if it is present, and store them in the specified arrays.

Returns: 0 on success; -1 on failure.

If `palette` is NULL, no palette is loaded, even if there is one stored with the image. If the image in the file is compressed, `DFR8getimage` automatically decompresses it.

If `DFR8getdims` has not been called, `DFR8getimage` finds the next image in the same way that `DFR8getdims` does.

NOTE: The variables `bufwidth` and `bufheight` give the number of columns and rows, respectively, in the array which you've allocated in memory to store the image. The image may actually be smaller than the allocated space.

NOTE: The order in which you declare dimensions is different between C and FORTRAN. Ordering varies because FORTRAN arrays are stored in column-major order, while C arrays are stored in row-major order. (*Row-major order* implies that the horizontal coordinate varies fastest). When `DFR8putimage` writes an image to a file, it assumes row-major order. The FORTRAN declaration that causes an image to be stored in this way must have the width as its first dimension and the height as its second dimension. To take this into account as you build your image in your program, you need to build the image "on its side."

Figure 2.3 Storing Multiple RIS8s in a Single File

FORTRAN:

```
INTEGER        DFR8setpalette, DFR8putimage, DFR8addimage
CHARACTER*1    palA(768), palB(768)
CHARACTER*1    pic1(1200,800), pic2(1200,800)
CHARACTER*1    pic3(1200,800), pic4(1200,800)
INTEGER        ret, DFTAG_RLE

PARAMETER (DFTAG_RLE = 11)

ret = DFR8setpalette(palA)
ret = DFR8putimage('myfile',pic1,1200,800,DFTAG_RLE)
ret = DFR8addimage('myfile',pic2,1200,800,DFTAG_RLE)
ret = DFR8setpalette(palB)
ret = DFR8addimage('myfile',pic3,1200,800,0)
ret = DFR8addimage('myfile',pic4,1200,800,0)
  .
  .
  .
```

Reading 8-Bit Raster Images from a File

The two routines, `DFR8getdims` and `DFR8getimage`, are sufficient to read raster images from a file. If enough is known about the images and palettes, only the latter routine is needed.

DFR8getdims

FORTRAN:

```
INTEGER FUNCTION DFR8getdims(filename,width,height,ispalette)

CHARACTER*(*)  filename       -  name of file with RIS8 image
INTEGER        width, height  -  dimensions of next image in file
INTEGER        ispalette      -  1 if there is a palette, else 0
```

C:

```
int DFR8getdims(filename,width,height,ispalette)

char     *filename;        /* name of file with RIS8 image */
int32    *width, *height;  /* dimensions of next image in
                              file */
int      *ispalette;       /* 1 if there is a palette, else
                              0 */
```

Purpose: To open the file with name `filename`, find the next image, retrieve the dimensions of the image in `width` and `height`, and tell, via `ispalette`, whether there is a palette associated with the image.

Returns: 0 on success; -1 on failure.

If the file is being opened for the first time, `DFR8getdims` returns information about the first image in the file. If an image has already been read, `DFR8getdims` finds the next image. Thus, images are read in the same order in which they were written to the file.

Normally, `DFR8getdims` is called before `DFR8getimage` so that if necessary, space allocations for the image and palette can be checked, and the dimensions can be verified. If this information is already known, `DFR8getdims` need not be called.

```
int32   width, height;         /* dimensions of the image */
int     compress;              /* type of compression to use, if
   any */
```

Purpose: To append to the file the RIS8 for the image.

Returns: 0 on success; -1 on failure.

In all other respects, `DFR8addimage` is functionally equivalent to `DFR8putimage`.

Example: Writing a Palette and an Image in RIS8 Format

Figure 2.2 demonstrates in FORTRAN how a palette stored in the array `colors` and a raw image stored in the 400 x 600 array `picture` (height=400, width=600) are written to a file in RIS8 format.

Figure 2.2 Storing an RIS8

```
FORTRAN:
INTEGER        DFR8setpalette, DFR8putimage
CHARACTER*1    colors(768)
CHARACTER*1    picture(600,400)
INTEGER        ret

ret = DFR8setpalette(colors)
ret = DFR8putimage('herfile.hdf',picture,600,400,0)

if (ret .ne. 0)
write(*,*) 'Error writing image to myfile.hdf.'
   .
   .
   .
```

Remarks:

- The RIS8 with this palette and image is stored as the *first* image in '`herfile.hdf`'. Note that if something already existed in this file, it will be lost, because `DFR8putimage` recreates the file. If you simply want to append an image to the file, use `DFR8addimage`.
- The image is not compressed in the file.
- The palette is assumed to be organized in the manner described earlier: R, G, B (first color); R, G, B (second color); and so forth.

Example: Writing a Series of RIS8 Images

Figure 2.3 illustrates a series of FORTRAN calls in which four 800 x 1200 (height=800; width=1200) images are written to the same file. The first two use palette `palA` and are compressed using the run length encoding technique; the third and fourth use palette `palB` and are not compressed.

```
C:
int DFR8putimage(filename, image, width, height, compress)

char     *filename;      /* name of file to store RIS8 in */
int32    width, height;  /* dimensions of image */
char     *image;         /* array with image to put in file  */
int      compress;       /* type of compression to use, if any */
```

Purpose: To write out the RIS8 for the image as the first image in the file.

Returns: 0 on success; -1 on failure.

If before the call to `DFR8putimage` there was other information in the file, the function overwrites that information.

The argument `compress` identifies the scheme to be used for compressing the data, if any. Refer to Table 2.2 for valid values of `compress`.

If IMCOMP compression is used, the image **must** include a palette. (See the discussion of 8-bit compression schemes in the section, “Compression Schemes.”)

NOTE: `DFR8addimage` (see below) writes an image to a file by appending it, rather than overwriting it.

NOTE: In FORTRAN, the dimensions of the array `image` must be the same as the dimensions of the image itself.

NOTE: The order in which you declare dimensions is different between C and FORTRAN. Ordering varies because FORTRAN arrays are stored in column-major order, while C arrays are stored in row-major order. (*Row-major order* implies that the horizontal coordinate varies fastest). When `DFR8putimage` writes an image to a file, it assumes row-major order. The FORTRAN declaration that causes an image to be stored in this way must have the width as its first dimension and the height as its second dimension. To take this into account as you build your image in your program, you need to build the image “on its side.”

DFR8addimage

```
FORTRAN:
INTEGER FUNCTION DFR8addimage(filename,image,width,height, compress)

CHARACTER*(*)  filename       -   name of file to add RIS8 to
CHARACTER      image(width,height)    -     array holding image
                                  to be added to file
INTEGER        width, height  -   dimensions of the image
INTEGER        compress       -   type of compression to use, if
                                  any
```

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

char   *filename;             /*  name of file to add RIS8 to */
char   image[height][width];  /* array holding image to add to
   file */
```

A Note About IMCOMP

IMCOMP should be used with caution if you are concerned about losing information in your image. *IMCOMP compression* first breaks an image into 4 x 4 arrays of pixels, then for each array chooses two colors to distribute in the array. (These two colors are added to a 256-color palette that IMCOMP compression builds. This new palette is based on, but different from, the original palette assigned.) Each of the 16 pixels in the 4 x 4 array can now be represented by one bit (0=first color; 1=second color). In addition to these sixteen bits, there are two bytes that give the palette locations of the two colors that were assigned to the 4 x 4 array.

Since each 4 x 4 array uses only 4 bytes, IMCOMP stores an image at a cost of 2 bits per pixel, which is 25% of the original storage requirement for the 8-bit image. The drawback of this savings is loss of information—only two colors are allowed to occupy each 4 X 4 array of pixels—whereas in the original image, 16 colors could occupy the same space. For many images this cost is bearable and hardly noticeable, but for some images, the results can be totally unrecognizable.

Also note that IMCOMP is dependent on the existence of a palette. If you are going to use IMCOMP, you **must** include a palette with your image.

Writing 8-Bit Raster Images to a File

DFR8setpalette

FORTRAN:

```
INTEGER FUNCTION DFR8setpalette(palette)

CHARACTER*1  palette(768)        -   palette to go with image
```

C:

```
int DFR8setpalette(palette)

char     palette[768];       /* palette to go with image */
```

Purpose: To indicate what palette, if any, is to be used for subsequent images.

Returns: 0 on success; -1 on failure.

The palette that is set here continues as the default palette until it is changed by a new call to the routine.

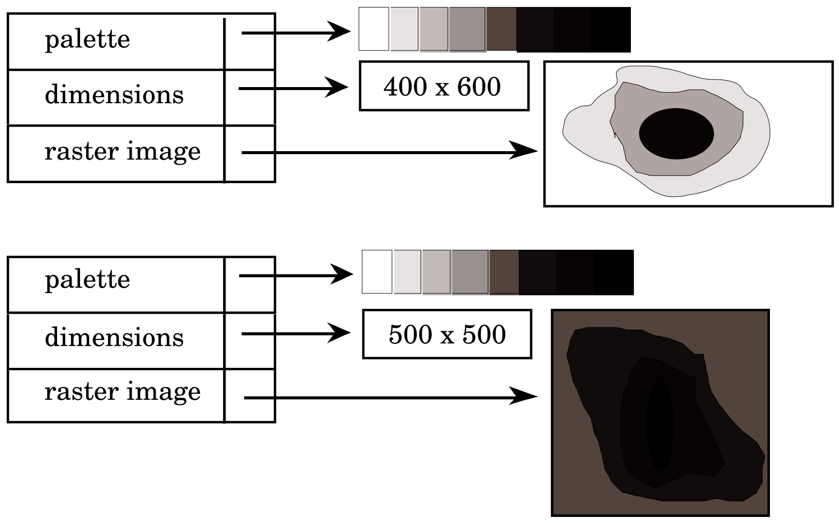
DFR8putimage

FORTRAN:

```
INTEGER FUNCTION DFR8putimage(filename, image, width, height, com-
press)

CHARACTER*(*)filename      -   name of file to store RIS8 in
INTEGER      width, height -   dimensions of image
CHARACTER    image(width,height)     - array holding image to be
                                       put in file
INTEGER      compress      -   type of compression to use, if
             any
```

Figure 2.1 Two Raster Image Sets in an HDF File



Compression Schemes

A *compression scheme* indicates if and how an image is compressed. Compression schemes currently supported by NCSA HDF are *run length encoding* and *IMCOMP*. The value of the integer argument `compress` in `DFR8putimage` and `DFR8addimage` determines which scheme, if any, is to be used, as shown in
Table 2.2.

Table 2.2 Compression Scheme Codes

Value	**Compression Scheme**
0	none
DFTAG_RLE	run length encoding (RLE)
DFTAG_IMCOMP	IMCOMP

The HDF tags `DFTAG_RLE` and `DFTAG_IMCOMP` are defined as the values 11 and 12, respectively, in the file `'df.h'`. You can avoid using numbers for compression codes if you include this file in your program.

A Note About RLE

The *run length encoding* (RLE) method used in HDF works as follows: Each sequence of pixels begins with a *count byte.* The low seven bits of the count byte indicate the number of bytes in the sequence (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).

The amount of space saved by RLE depends upon how much repetition there is among the pixels in the rows. (Pixels are stored in rows.) If there is a great deal of repetition, much space is saved; if there is little repetition, the savings can be very small. In the worst case—when every pixel is different from the one that precedes it—an extra byte is added for every 127 bytes in the image.

tion is stored in the same file as the actual images, the software does not have to search elsewhere for this pertinent information. More importantly, you may be spared from having to supply the information. This reduction in the need to coordinate disparate pieces of information about a raster image can make the job of creating and running image-processing programs significantly easier.

8-Bit Raster Image Sets

The phrase, *8-bit raster image set* (RIS8), refers to the set of tags and associated information required to store an 8-bit raster image in an HDF file. An RIS8 contains at least the first three of the following components and may also contain a palette:

- An *image*—here, a two-dimensional array of 8-bit numbers, one for each pixel in the raster image, where pixel values range from 0 to 255 (Pixel values indicate to the hardware which colors to use when drawing the corresponding pixels on the screen.)

- *Dimensions*—two values that represent the x and y dimensions of the image, respectively

- A *compression scheme*—a code that indicates if and how the image was compressed (See the following section, “Compression Schemes.”)

- A *palette*—a lookup table with 256 entries that tells the color to associate with each of the 256 possible pixel values (Each entry in the palette is chosen from a master palette of 2^{24} RGB colors. Each palette entry consists of three bytes, one each for red, green, and blue. The first three bytes represent the R, G, and B values of the first color in the palette; the next three the R, G, and B values of the second color; and so forth. The total size of a palette is 768 bytes.)

An example of an HDF file with two raster image sets is illustrated in Figure 2.1.

Table 2.1 Raster Image Set Routines in the HDF Library (Continued)

Long Name	Short Name	Function
DFR8getimage	**d8gimg**	retrieves the image and any associated palette, and stores them in arrays.
DFR8readref	**d8rref**	sets the reference number of the image to get next
DFR8restart	**d8first**	gets the next `get` command to read from the first RIS8 in the file, rather than the next.
DFR8nimages		counts the number of images stored in the file (FORTRAN version currently not available)
DFR8lastref		returns reference number of last RIS8 read or written
DF24setil	**d2setil**	sets the interlace to be used when writing out the RIS24 for the image.
DF24addimage	**d2iaimg**	appends the RIS24 for the image to the file.
DF24getdims	**d2igdims**	retrieves the dimensions and interlace of the image.
DF24getimage	**d2igimg**	retrieves the image and stores it in an array.
DF24readref	**d2rref**	sets the reference number of the image to get next
DFR24restart	**d8first**	causes the next `get` command to read from the first RIS24 in the file, rather than the next one.
DF24reqil	**d2reqil**	specifies an interlace to be used in place of the interlace indicated in the file when the next raster image is read.

Reasons to Use Raster Image Sets

When raster images are stored in the form of HDF raster image sets, it becomes possible to use a variety of software tools for displaying and manipulating them. NCSA Image, for instance, can operate directly on images stored in HDF raster image format. Other software can display raster images in HDF format on a variety of different machines.

The use of raster image sets in HDF files makes it easier for programs to handle raster images. When palette, dimension, or compression informa-

Chapter Overview

This chapter discusses the purposes and use of raster image sets, which allow you to store an image, together with its dimensions and a palette, in an HDF file. This chapter specifically introduces and describes the two raster image set interfaces currently contained in the HDF library: RIS8 and RIS24.

Header Files

The header file `dfrig.h` contains the declarations and definitions that are used by the routines listed in this chapter. This file can, if needed, be included with your C source code, and in some cases also with FORTRAN code.

Raster Image Sets

A *raster image set* (RIS) is a set of tags and associated information required to store an image in an HDF file. In HDF, 8-bit raster image sets (RIS8) are used to store 8-bit raster images, and 24-bit raster image sets (RIS24) are used to store 24-bit raster images.

The HDF library currently contains routines for storing raw raster images in RIS8 or RIS24 format and for retrieving raster images from files containing raster image sets. These routines are callable from C and FORTRAN programs that have access to the library. They work on several computers, including Cray systems running UNICOS, Alliant, Vax, Sun, and Macintosh models.

Table 2.1 lists the long and short names and the functions of the RIS8 and RIS24 routines currently contained in the HDF library. The following sections provide descriptions and examples of these calling routines.

Table 2.1 Raster Image Set Routines in the HDF Library

Long Name	Short Name	Function
DFR8setpalette	**d8spal**	sets the default palette to be used for subsequent images.
DFR8putimage	**d8pimg**	writes out the RIS8 for the image as the first image in the file.
DFR8addimage	**d8aimg**	appends the RIS8 for the image to the file.
DFR8getdims	**d8gdims**	retrieves the dimensions of the image and indicates whether a palette is associated and stored with the image.

Chapter 2 Storing Raster Images