Chapter 8

General Raster Images (GR API)

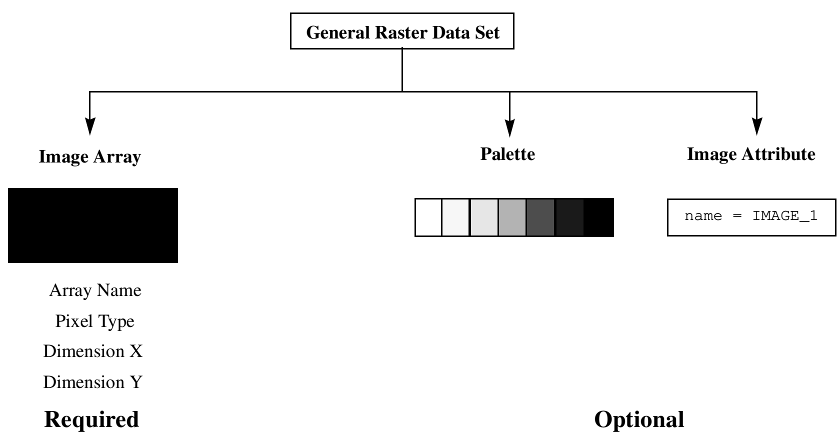
8.1 Chapter Overview

This chapter describes the general raster (GR) data model and the interface routines used to manipulate general raster data objects. As the general raster data model is designed to provide a more flexible means of storing raster image data than the RIS8 and RIS24 models, a brief comparison of the GR model with the RIS8 and RIS24 data models is also provided.

8.2 The General Raster Data Model

HDF users familiar with the SD interface will find the general raster data model a simplified version of the SD scientific data set model, customized to accommodate image data storage and manipulation. The raster image data is stored in a two-dimensional array and attributes can be created for the image, the file or both. Palettes can be created and attached to the image as well as compression method information. A fundamental difference between the SD scientific data model and the GR raster data model is the absence of customizable dimensional information in the GR data set. Refer to Figure 8a.

FIGURE 8a **General Raster Image Data Set Contents**



8.2.1 Required General Raster Objects

Every general raster data set must contain three objects. These objects are the ***image array*** , the ***image array name*** and ***pixel type*** . Required objects are automatically created from the information provided at the time the data set is defined.

8.2.1.1 Image Array

An ***image array*** is the two-dimensional structure used to store the image's pixel data.

8.2.1.2 Image Array Name

Each image array has an ***image name*** consisting of a string of case-sensitive alphanumeric characters. The name must be provided by the calling program; the GR interface does not provide one by default if one isn't specified. Names are assigned when the data set is created and cannot be changed afterwards. Image array names do not have to be unique within a file, but if they are not it can be difficult to distinguish between the general raster data sets in the file.

8.2.1.3 Pixel Type

Another fundamental difference between the SD SDS model and the GR data model is that a GR image data array is defined by data type of its elements and the number of components in each element rather than only the data type. This is because each element in an image array corresponds to one pixel and each element of pixel data can consist of a variable number of color component values. (Red-Blue-Green or RGB, Cyan-Magenta-Yellow-Black or CMYK, etc.) These color component values can be represented by different methods and data lengths (8-bit lookup table or 24-bit direct representation, graphically depicted in Figure 6a in Chapter 6 and Figure 7b in Chapter 7 respectively). The data type of array elements and the number of components in each element are collectively referred to as the ***pixel type***.

Data Type

As of HDF version 4.1r1, the general raster data model supports any data type.

Number of Components

Pixel elements can be comprised of any number of components.

8.2.1.4 Dimensions

Image array ***dimensions*** specify the size of an image array. There are no routines in the GR library that allow the HDF user to add attributes to a dimension or change the scale, as is possible in the SD interface.

8.2.2 Optional General Raster Objects

There are two types of optional objects available for inclusion in a general raster data set: ***palettes*** and ***attributes***. These objects are only created when specifically requested by the calling program; the GR interface doesn't provide predefined palettes or attributes.

Palettes

Palettes are lookup tables attached to images for the purpose of the determining a set of color values for each pixel value in the image array. The GR interface provides similar capabilities for storing and manipulation palettes as the DFP interface described in Chapter 9, titled *Palettes (DFP API)*. However, the DFP interface is restricted to single-file operations while the GR interface allows multifile palette operations. Eventually, all palette manipulation functionality will reside only within the GR interface, but for the time being the single-file DFP routines are fully compatible with palettes created with the GR palette routines. The GR palette routines are described in Section 8.9 on page 233.

Attributes

Attributes are defined by the calling program to contain auxiliary information about a file, image or both. They are described in Chapter 3, titled *Scientific Data Sets (SD API)*. The GR implementation of attributes is covered in Section 8.8 on page 227.

8.3 The General Raster API

The GR interface consists of routines for storing, retrieving and manipulating the data in general raster data sets.

8.3.1 GR API Routines

All C routine names in the general raster interface have the prefix “GR” and the equivalent Fortran-77 routine names are prefaced by "mg". All GR routines are classifiable within one of the following categories:

- ***Access routines*** initialize and terminate access to the GR interface and general raster data sets.
- ***Read/write routines*** modify the data and metadata contained in a general raster data set.
- ***Maintenance routines*** create and destroy the data and metadata contained in a general raster data set and modify global settings governing the format of the stored data.
- ***Inquiry routines*** return information about data contained in a general raster data set.

The GR function calls are listed in the following table and described further in the *HDF Reference Guide*.

TABLE 8A **GR Library Routines**

Purpose	Routine Name		Description
	C	Fortran-77	
Access	GRstart	mgstart	Initializes the GR interface for a given data file.
	GRend	mgend	Terminates access to the file initialized by **GRstart**.
	GRselect	mgselct	Selects the data set to perform operations on.
	GRendaccess	mgendac	Terminates access to the data set selected by **GRselect** or **GRcreate**.
Read/write	GRreadimage	mgrdimg/ mgrcimg	Reads image data from a general raster data set.
	GRwriteimage	mgwrimg/ mgwcimg	Writes image data to a general raster data set.
	GRidtoref	mgid2ref	Maps an general raster data set identifier to a reference number.
	GRluttoref	None	Returns the reference number for the specified palette..
	GRreftoindex	mgr2idx	Maps the reference number of a general raster data set to a data set index and returns the index.
	GRnametoindex	mgn2ndx	Maps a raster image name to an index and returns the index.
	GRreadlut	mgrdlut/ mgrclut	Reads palette data from a general raster data set.
	GRwritelut	mgwrlut/ mgwclut	Writes palette data to a general raster data set.
	GRsetattr	mgsnatt/ mgscatt	Writes the attribute of an object to a general raster data set.
	GRgetattr	mggnatt/ mggcatt	Reads the attribute of an object from a general raster data set.

Maintenance	GRcreate	mgcreat	Creates a new general raster data set.
	GRreqlutil	mgrltil	Sets the interlace mode for the next palette read from a general raster data set.
	GRreqimageil	mgrimil	Sets the interlace mode for the next image read from a general raster data set.
	GRgetlutid	mggltid	Allocates a palette id to a general raster data set.
	GRsetexternalfile	mgsxfil	Specifies that the image data of a general raster data set is a special element of an external element.
	GRsetaccesstype	mgsactp	Sets the access to a general raster data set to be either parallel or serial.
	GRsetcompress	mgscomp	Makes the image data of a general raster data set a compressed special element.
Inquiry	GRfileinfo	mgfinfo	Returns global information on the specified GR data set.
	GRgetiminfo	mggiinf	Returns information on the specified general raster data set.
	GRgetlutinfo	mgglinf	Returns information on a given palette.
	GRattrinfo	mgatinf	Returns attribute information on a given object.
	GRfindattr	mgfndat	Returns the index of an attribute with the given name.

8.4 Programming Model for the GR Interface

As with the SD interface, the GR interface relies on the calling program to initiate and terminate access to files and data sets to support multifile access. The GR programming model for accessing a general raster data set is as follows:

1. Open a file by obtaining a file id from a file name.
2. Initialize the GR interface and obtain a general raster interface id.
3. Open an existing general raster data set by obtaining a data set id from the interface id and data set index **OR** create a new general raster data set by obtaining a data set id from the data set name and data type.
4. Perform desired operations on the data set.
5. Dispose of the raster image id.
6. Terminate access to the GR interface by disposing of the interface id.
7. Close the file.

To access a single raster image data set in an HDF file, the calling program must contain the following calls:

C:

```
file_id = Hopen(filename, access_mode, number_of_desc);
gr_id = GRstart(file_id);
ri_id = GRselect(gr_id, gr_index);
<Optional operations>
status = GRendaccess(ri_id);
status = GRend(gr_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, access_mode, number_of_desc)
gr_id = mgstart(file_id)
ri_id = mgselct(gr_id, gr_index)
<Optional operations>
status = mgendac(ri_id)
status = mgend(gr_id)
status = hclose(file_id)
```

To access several files at the same time, a calling program must obtain a separate file id for each file to be opened. Similarly, to access more than one general raster data set a calling program must obtain a separate interface id for each data set. For example, to open two data sets stored in two files a program would execute the following series of function calls:

C:

```
file_id_1 = Hopen(filename_1, access_mode, number_of_desc);
gr_id_1 = GRstart(file_id_1);
ri_id_1 = GRselect(gr_id_1, gr_index_1);
file_id_2 = Hopen(filename_2, access_mode, number_of_desc);
gr_id_2 = GRstart(file_id_2);
ri_id_2 = GRselect(gr_id_2, gr_index_2);
<Optional operations>
status = GRendaccess(ri_id_1);
status = GRend(gr_id_1);
status = Hclose(file_id_1);
status = GRendaccess(ri_id_2);
status = GRend(gr_id_2);
status = Hclose(file_id_2);
```

FORTRAN:

```
file_id_1 = hopen(filename_1, access_mode, number_of_desc)
gr_id_1 = mgstart(file_id_1)
ri_id_1 = mgselct(gr_id_1, gr_index_1)
file_id_2 = hopen(filename_2, access_mode, number_of_desc)
gr_id_2 = mgstart(file_id_2)
ri_id_2 = mgselct(gr_id_2, gr_index_2)
<Optional operations>
status = mgendac(ri_id_1)
status = mgend(gr_id_1)
status = hclose(file_id_1)
status = mgendac(ri_id_2)
status = mgend(gr_id_2)
status = hclose(file_id_2)
```

Because every file and general raster data set is assigned its own identifier, the order in which files and data sets are accessed is very flexible as long as all file and general raster data set ids are individually discarded before the end of the calling program.

8.4.1 Accessing Files and Images: GRstart and GRselect

In the GR interface, **Hopen** is used to open files. This differs from the SD programming model where **SDstart** is used for this purpose. For information on the use of **Hopen** refer to Chapter 2, titled *HDF Fundamentals* and for information on **SDstart** refer to Chapter 3, titled *Scientific Data Sets (SD API)*.

GRstart initializes the GR interface and must be called once after **Hopen** and before any other GR routines are called. It takes one argument; the `file_id` returned by **Hopen** and returns the interface id `gr_id`.

GRselect specifies the given image as the current image to be accessed. It takes two arguments: the `gr_id` returned by **GRstart** and `gr_index` and returns the raster image id `ri_id`. The argument `gr_index` is the position of the data set relative to the beginning of the file. The argument `gr_index` is zero-based, meaning that the index of first image in the file is 0.

The parameters for **GRstart** and **GRselect** are further defined below. (See Table 8B.)

8.4.2 Terminating Access to Files and Images: GRendaccess and GRend

GRendaccess disposes of the open raster image id `ri_id` and terminates access to the data set initiated by the corresponding call to **GRselect**. The calling program must make one **GRendaccess** call for every **GRselect** call made during its execution. Failing to call **GRendaccess** for each call to **GRselect** may result in a loss of data.

GRend disposes of the general raster data set id `gr_id` and terminates the access to the GR interface initiated by the corresponding call to **GRstart**. The calling program must make one **GRend** call for every **GRstart** call made during its execution and failing to call **GRend** for each **GRstart** may result in a loss of data. The parameters for **GRendaccess** and **GRend** are listed and defined in the following table.

TABLE 8B **GRstart, GRselect, GRend and GRendaccess Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRstart (**mgstart**)	file_id	int32	integer	File identifier.
GRselect (**mgselct**)	gr_id	int32	integer	General raster data set identifier.
	gr_index	int32	integer	Position of the general raster data set within the file.
GRend (**mgend**)	gr_id	int32	integer	General raster data set identifier.
GRendaccess (**mgendac**)	ri_id	int32	integer	Image array identifier.

8.5 Creating and Writing General Raster Images

This section describes the routines needed to create and write simple general raster data sets. A "simple" general raster data set is defined here as one with no attributes.

Creating a general raster data set and writing data to it are separate operations in the GR programming model. Once an array is defined it is ready to store data. No definitions are retained about the size, contents or number of components of a general raster data set from one data set to the next or from one file to the next.

8.5.1 Creating General Raster Images: GRcreate

Creating a simple general raster data set requires the following steps:

1. Open a file and initialize the GR interface.
2. Define the characteristics of the general raster data set.
3. Perform optional operations on the data set.
4. Terminate access to the data set.
5. Terminate access to the GR interface and close the file.

To create the data set, the calling program must contain the following sequence of function calls:

C:
```
file_id = Hopen(filename, access_mode, number_of_desc);
gr_id = GRstart(file_id);
ri_id = GRcreate(gr_id, name, ncomp, number_type, il,
        dim_sizes);
<Optional operations>
status = GRendaccess(ri_id);
status = GRend(gr_id);
status = Hclose(file_id);
```

FORTRAN:
```
file_id = hopen(filename, access_mode, number_of_desc)
gr_id = mgstart(filename, access_mode)
ri_id = mgcreat(gr_id, name, ncomp, number_type, il,
        dim_sizes);
<Optional operations>
```

```
status = mgendac(ri_id)
status = mgend(gr_id)
status = hclose(file_id)
```

GRcreate defines a new general raster data set using the arguments `name`, `ncomp`, `number_type`, `il` and `dim_sizes`. Once a data set is created, you cannot change its name, data type, dimension or number of components. **GRcreate** does not actually perform the write; this occurs only when **GRendaccess** is called.

The maximum length of a data set name is defined by `MAX_GR_NAME` and the maximum number of components of an image array is defined by `MAX_VAR_DIMS`. Both are defined in the "hlimits.h" header file.

When creating a general raster data set, it is necessary to specify the data type of the image array data. The "hntdefs.h" header file contains definitions of all valid data types, which are described in Chapter 2, titled *HDF Fundamentals*. The GR interface supports all regular HDF data types. However, it does not include the support for "n-bit" integers which is included in the SD interface.

TABLE 8C **GRcreate Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRcreate (mgcreat)	gr_id	int32	integer	General raster data set identifier.
	name	char *	character* (*)	Name of the image.
	ncomp	int32	integer	Number of components in each element of the data set.
	data_type	int32	integer	Data type of the data set.
	interlace_mode	int32	integer	Interlace mode to be used when writing to the data set
	dim_sizes	int32 [2]	integer (2)	Array defining the size of both dimensions.

8.5.2 Setting the Interlace Mode

During the process of reading a palette or an image, the interlace mode can be set to either MFGR_INTERLACE_PIXEL, MFGR_INTERLACE_LINE or MFGR_INTERLACE_COMPONENT. These definitions respectively correspond to pixel interlacing, line interlacing and component interlacing. The first two interlacing modes are illustrated for the instance of 24-bit pixel representation in Figure 7c of Chapter 7, titled *24-bit Raster Images (DF24 API)*. Component interlacing, as the name implies, describes interlacing raster data by color component.

EXAMPLE 1. **Creating and Accessing a General Raster Data Set**

In these examples an empty array is created, or one that has been defined but not yet initialized with data.

C:

```
#include "hdf.h"
#include "mfgr.h"

#define X_LENGTH 5
#define Y_LENGTH 10

main( )
{

int32 gr_id, ri_id, file_id, status;
int32 dimsizes[2], ncomp, il;

/* Create and open the file. */
file_id = Hopen("Example1.hdf", DFACC_CREATE, 0);

/* Initiate the GR interface. */
gr_id = GRstart(file_id);

/* Define the number of components and dimensions of the image. */
ncomp = 2;
il = MFGR_INTERLACE_PIXEL;
dimsizes[0] = X_LENGTH;
dimsizes[1] = Y_LENGTH;

/* Create the image array. */
ri_id = GRcreate(gr_id, "Image_array_1", ncomp, DFNT_INT16, il,
                dimsizes);

/* Terminate access to the image array. */
status = GRendaccess(gr_id);

/* Terminate access to the GR interface and close the file */
status = GRend(ri_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM CREATE IMAGE ARRAY

      integer*4 gr_id, ri_id, file_id, dimsizes(2), ncomp, il
      integer mgstart, mgcreat, mgendac, mgend, hopen, hclose
      integer*4 X_LENGTH, Y_LENGTH, status
      parameter (X_LENGTH = 5, Y_LENGTH = 10,
     +           MFGR_INTERLACE_PIXEL = 0)
C     DFACC_CREATE and DFNT_INT16 are defined in hdf.h.
      integer*4 DFACC_CREATE, DFNT_INT16
      parameter (DFACC_CREATE = 4, DFNT_INT16 = 22)

C     Create and open the file.
      file_id = hopen("Example1.hdf", DFACC_CREATE, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Define the number of components and dimensions of the image
C     array.
```

```
      ncomp = 2
      il = MFGR_INTERLACE_PIXEL
      dimsizes(1) = Y_LENGTH
      dimsizes(2) = X_LENGTH

C     Create the image array.
      ri_id = mgcreat(gr_id, 'Ex_array_1', ncomp, DFNT_INT16, il,
     +                dimsizes)

C     Terminate access to the image array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface and close the file.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)
      end
```

8.5.3 Writing General Raster Images: GRwriteimage

GRwriteimage is the routine used to either completely or partially fill an n-component image array. It can also skip a specified number of image array elements between write operations along each dimension.

Writing data to an image array involves the following steps:

1. Open a file and initialize the GR interface.
2. Select or create a data set.
3. Write data to the image array.
4. Terminate access to the data set.
5. Terminate access to the file and close the file.

The calling program must contain the following sequence of calls:

C:

```
file_id = Hopen(filename, access_mode, number_of_desc);
gr_id = GRstart(file_id);
ri_id = GRcreate(gr_id, name, ncomp, number_type, il,
dim_sizes);
status = GRwriteimage(ri_id, start, stride, edge, data);
status = GRendaccess(gr_id);
status = GRend(ri_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, access_mode, number_of_desc)
gr_id = mgstart(file_id)
ri_id = mgcreat(gr_id, name, ncomp, number_type, il,
dim_sizes);
status = mgwrimg(ri_id, start, stride, edge, data)
status = mgendac(ri_id)
status = mgend(gr_id)
status = hclose(file_id)
```

As with SD arrays, subsets of general raster images can be written. (In n-dimensional SD arrays these two-dimensional subsets are referred to as “slabs”.) A subset is specified by giving an array consisting of the coordinate location of the origin vertex and an array consisting of lengths of each of the two dimensions.

GRwriteimage takes five arguments: `ri_id`, `start`, `stride`, `edge`, and `data`. The `ri_id` argument is the raster image id returned by **GRcreate** or **GRselect**. The arguments `start`, `stride`, and `edge` respectively describe the initial location in the image for the write operation, the number of locations the current array location will be moved forward after each write, and the length of each dimension of the subset to be written. If the image array is smaller than the `data` argument array, the amount of data written will be truncated to the size of the image array. These concepts are explained further in Chapter 3, titled *Scientific Data Sets (SD API)*.

There are two Fortran-77 versions of this routine: **mgwrimg** and **mgwcimg**. The **mgwrimg** routine writes buffered numeric data and the **mgwcimg** routine writes buffered character data.

The parameters for **GRwriteimage** are described in the following table. Note that, because there are two Fortran-77 versions of **GRwriteimage**, there are correspondingly two entries in the "Data Type" field of the *data* parameter.

TABLE 8D

GRwriteimage Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRwriteimage (mgwrimg/ mgwcimg)	ri_id	int32	integer	Image array identifier returned by **GRcreate**.
	start	int32 [2]	integer (2)	Array containing the x,y-coordinate location the write will start for each dimension.
	stride	int32 [2]	integer (2)	Array containing the number of data locations the current location is to be moved forward before the next write.
	edge	int32 [2]	integer (2)	Array containing the number of data elements that will be written along each dimension.
	data	VOIDP	<valid numeric data type>	Buffer array containing the data to be written.

EXAMPLE 2.

Creating and Writing an Image

These examples use **GRcreate** and **GRwriteimage** under the C interface and **mgcreat** and **mgwrimg** under the Fortran-77 interface.

C:

```
#include "hdf.h"
#include "mfgr.h"

#define X_LENGTH 15
#define Y_LENGTH 10

main( )
{
int32 gr_id, ri_id, file_id, status;
int32 dims[2], start[2], edges[2], ncomp, il;
int16 image_data[Y_LENGTH][X_LENGTH][2], i, j;

/* Create and open the file. */
file_id = Hopen("Example2.hdf", DFACC_CREATE, 0);

/* Initiate the GR interface. */
gr_id = GRstart(file_id);
```

```
/* Define the number of components and dimensions of the image. */
ncomp = 2;
il = MFGR_INTERLACE_PIXEL;
dims[0] = X_LENGTH;
dims[1] = Y_LENGTH;

/* Create the array. */
ri_id = GRcreate(gr_id, “Ex_array_2”, ncomp, DFNT_INT16, il, dims);

/* Fill the stored-data array with values. */
for (j = 0; j < Y_LENGTH; j++) {
    for (i = 0; i < X_LENGTH; i++) {
        image_data[j][i][0] = (i + j) + 1;
        image_data[j][i][1] = (i + j) + 1;
    }
    }

/* Define the location, pattern, and size of the data set */
for (i = 0; i < 2; i++) {
    start[i] = 0;
    edges[i] = dims[i];
}

/* Write the stored data to the image array. */
status = GRwriteimage(ri_id, start, NULL, edges, (VOIDP)image_data);

/* Terminate access to the array. */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface. */
status = GRend(gr_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM WRITE ARRAY

      integer*4 ri_id, gr_id, file_id, ncomp, il
      integer dims(2), start(2), edges(2), stride(2)
      integer i, j, status
      integer mgstart, mgcreat, mgwrimg, mgendac, mgend
      integer hopen, hclose
      integer*4 DFACC_CREATE, DFNT_INT16
      integer*4 X_LENGTH, Y_LENGTH
      parameter (DFACC_CREATE = 4, DFNT_INT16 = 22, X_LENGTH = 15,
     +           Y_LENGTH = 10, MFGR_INTERLACE_PIXEL = 0)
      integer*2 image_data(2, X_LENGTH, Y_LENGTH)

C     Create and open the file.
      file_id = hopen(‘Example2.hdf’, DFACC_CREATE, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Define the number of components and dimensions of the image.
      ncomp = 2
      il = MFGR_INTERLACE_PIXEL
```

```
      dims(1) = X_LENGTH
      dims(2) = Y_LENGTH

C     Create the data set.
      ri_id = mgcreat(gr_id, 'Ex_array_2', ncomp, DFNT_INT16, il, dims)

C     Fill the stored-data array with values.
      do 20 j = 1, Y_LENGTH
         do 10 i = 1, X_LENGTH
            image_data(1, i, j) = i + j - 1
            image_data(2, i, j) = i + j - 1
10       continue
20    continue

C     Define the location, pattern, and size of the data set
C     that will be written to.
      start(1) = 0
      start(2) = 0
      edges(1) = X_LENGTH
      edges(2) = Y_LENGTH
      stride(1) = 1
      stride(2) = 1

C     Write the stored data to the image array.
      status = mgwrimg(ri_id, start, stride, edges, image_data)

C     Terminate access to the array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

      end
```

8.6 Reading Data from an Image: GRreadimage

Reading one or more slabs from an image array involves the following steps:

1. Open the file and initialize the GR interface.
2. Select a data set.
3. Read data from the image array.
4. Terminate access to the data set.
5. Terminate access to the GR interface and close the file.

To read data from an image array, the calling program must contain the following function calls:

C:

```
file_id = Hopen(filename, access_mode, number_of_desc);
gr_id = GRstart(file_id);
ri_id = GRselect(gr_id, gr_index);
status = GRreadimage(ri_id, start, stride, edge, data);
status = GRendaccess(gr_id);
status = GRend(ri_id);
status = Hclose(file_id);
```

FORTRAN:

```
file_id = hopen(filename, access_mode, number_of_desc)
```

```
gr_id = mgstart(file_id)
ri_id = mgselct(gr_id, GRs_index)
status = mgrdimg(ri_id, start, stride, edge, data)
status = mgendac(gr_id)
status = mgend(ri_id)
status = hclose(file_id)
```

GRreadimage can be used to read either an entire image or a subset of the image. The `ri_id` argument is the raster image identifier returned by **GRselect**. As with **GRwriteimage**, the arguments `start`, `stride`, and `edge` respectively describe the starting location for the read operation, the number of locations the current image array location will be moved forward after each read, and the length of each dimension to be read. If the image array is smaller than the `data` argument array, the amount of data read will be limited to the maximum size of the image array.

There are two Fortran-77 versions of this routine: **mgrdimg** and **mgrcimg**. The **mgrdimg** routine reads numeric image data and **mgrcimg** reads character image data.

The parameters for **GRreadimage** are further defined below. (See Table 8E.)

8.6.1 Reading General Raster Images from an External File

Image data is read from an external or compressed file in the same way that it is read from a primary file.

8.6.2 Setting the Interlace Mode for a Palette or Image Read

There are two GR routines that deal with setting the interlace in raster image data sets: **GRreqlutil** and **GRreqimageil**. Refer to Section 8.5.2 on page 205 for a description of the GR interlace modes.

The **GRreqlutil** routine sets the interlace mode for the next palette read. It can be called at anytime before the read operation and takes two parameters, `ri_id` and `interlace_mode`. The `ri_id` parameter is the image array identifier returned by the **GRselect** routine and the `interlace_mode` is the interlace mode that will be in effect for the next image or palette read operation.

The **GRreqimageil** routine sets the interlace mode for the next image read. It also can be called at anytime before the read operation and takes the same two parameters as **GRreqlutil**. **GRreqlutil** and **GRreqimageil** may be called more than once; the interlace mode setting specified by the last call to either routine will be used for the next read operation.

Descriptions of both interlace mode routines are included in the following table. The GR palette read and write routines are covered in Section 8.9 on page 233. Note that, because there are two Fortran-77 versions of **GRreadimage**, there are correspondingly two entries in the "Data Type" field of the *data* parameter.

TABLE 8E **GRreadimage, GRreqlutil and GRreqimageil Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	

GRreadimage (mgrdimg/ mgrcimg)	ri_id	int32	integer	Image identifier.
	start	int32[2]	integer (2)	Array containing the starting read coordinates.
	stride	int32[2]	integer (2)	Array containing the number of data locations the current location is to be moved forward before the next read.
	edge	int32[2]	integer (2)	Array containing the number of data element that will be read along each dimension.
	data	VOIDP	<valid numeric data type>	Buffer for the returned image data.
GRreqlutil (mgrltil)	ri_id	int32	integer	Image identifier.
	interlace_mode	intn	integer	Interlace mode for the next palette read operation.
GRreqimageil (mgrimil)	ri_id	int32	integer	Image identifier
	interlace_mode	intn	integer	Interlace mode for the next image read operation.

EXAMPLE 3.

Reading an Image

When **GRreadimage** is used to read an entire image array, the coordinates for the start position must begin at 0 for each dimension (`start={0,0}`), the interval between each read must equal 1 for each dimension (`stride=NULL or stride={1,1}`), and the size of each dimension must equal the size of the array itself (`edge={dim_size_1, dim_size_2}`). The `data` buffer must have enough space allocated to hold the data. In these examples, the image created in Example 2 is read.

C:

```
#include "hdf.h"

#define X_LENGTH 15
#define Y_LENGTH 10

main( )
{
int32 gr_id, ri_id, file_id, status;
int32 start[2], edges[2], dims[2], nattrs;
int16 image_data[Y_LENGTH][X_LENGTH][2];

/* Open the file. */
file_id = Hopen("Example2.hdf", DFACC_RDONLY, 0);

/* Open the file and initiate the GR interface. */
gr_id = GRstart(file_id);

/* Select the first (and in this case, only) data set in the file. */
ri_id = GRselect(gr_id, 0);

/* Define the location, pattern, and size of the data to read. */
dims[0] = X_LENGTH;
dims[1] = Y_LENGTH;
start[0] = start[1] = 0;
edges[0] = dims[0];
edges[1] = dims[1];

/* Read the data in the image array. */
status = GRreadimage(ri_id, start, NULL, edges, (VOIDP)image_data);

/* Terminate access to the image array */
status = GRendaccess(ri_id);
```

```
/* Terminate access to the GR interface. */
status = GRend(gr_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
      PROGRAM READ ARRAY

      integer*4 gr_id, ri_id, file_id, status
      integer start(2), edge(2), stride(2)
      integer hopen, hclose, mgstart, mgselct, mgrdimg
      integer mgendac, mgend

C     DFACC_RDONLY is defined in hdf.h. MAX_NC_NAME and MAX_VAR_DIMS
C     are defined in netcdf.h.
      integer*4 DFACC_RDONLY, MAX_NC_NAME, MAX_VAR_DIMS
      integer*4 X_LENGTH, Y_LENGTH
      parameter (DFACC_RDONLY = 1, MAX_NC_NAME = 256,
     +           MAX_VAR_DIMS = 32, X_LENGTH = 15,
     +           Y_LENGTH = 10)
      integer*2 array_data(2, X_LENGTH, Y_LENGTH)
      integer dims(MAX_VAR_DIMS)

C     Open the file.
      file_id = hopen('Example2.hdf', DFACC_RDONLY, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Select the first (and in this case, only) data set in the file.
      ri_id = mgselct(gr_id, 0)

C     Define the location, pattern, and size of the data to read
C     from the data set.
      dims(1) = X_LENGTH
      dims(2) = Y_LENGTH
      start(1) = 0
      start(2) = 0
      stride(1) = 1
      stride(2) = 1
      edge(1) = dims(1)
      edge(2) = dims(2)

C     Read the array data set.
      status = mgrdimg(ri_id, start, stride, edge, array_data)

C     Terminate access to the image array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface and the file.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

      end
```

EXAMPLE 4.

Reading a Subset of an Image Array

The following examples show how the `start` and `edges` arguments of the **GRreadimage** function can be initialized so that a subset of the data set can be read.

C:

```
#include "hdf.h"

#define X_LENGTH 15
#define Y_LENGTH 10

main( )
{
int32 gr_id, ri_id, file_id;
int32 start[2], edges[2];
int16 all_image[2][Y_LENGTH][X_LENGTH];
int16 subset_image[2][7][3], status;

/* Open the file. */
file_id = Hopen("Example2.hdf", DFACC_RDONLY, 0);

/* Open the file for read-only access. */
gr_id = GRstart(file_id);

/* Select the first data set. */
ri_id = GRselect(gr_id, 0);

/* First, read the entire data set. */
start[0] = start[1] = 0;
edges[0] = X_LENGTH;
edges[1] = Y_LENGTH;
status = GRreadimage(ri_id, start, NULL, edges, (VOIDP)all_image);

/* Read a subset of the data set. */
start[0] = 1;
start[1] = 1;
edges[0] = 3;
edges[1] = 7;
status = GRreadimage(ri_id, start, NULL, edges, (VOIDP)subset_image);

/* Terminate access to the array. */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface. */
status = GRend(gr_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM READ SUBSET

      integer*4 gr_id, ri_id, file_id
      integer start(2), edges(2), stride(2), status
      integer mgstart, mgselct, mgrdimg, mgendac, mgend
      integer hopen, hclose
```

```
C     DFACC_RDONLY is defined in hdf.h.
      integer*4 DFACC_RDONLY
      integer*4 X_LENGTH, Y_LENGTH
      parameter (DFACC_RDONLY = 1, X_LENGTH = 15, Y_LENGTH = 10)
      integer*2 all_image(2, X_LENGTH, Y_LENGTH)
      integer*2 subset_image(2, 3, 7)

C     Open the file.
      file_id = hopen('Example2.hdf', DFACC_RDONLY, 0)

C     Initialize the GR interface.
      gr_id = mgstart(file_id)

C     Select the first data set.
      ri_id = mgselct(gr_id, 0)

C     Read the entire data set.
      start(1) = 0
      start(2) = 0
      edges(1) = X_LENGTH
      edges(2) = Y_LENGTH
      stride(1) = 1
      stride(2) = 1
      status = mgrdimg(ri_id, start, stride, edges, all_image)

C     Read a subset from the middle of the data set.
      start(1) = 1
      start(2) = 1
      edges(1) = 3
      edges(2) = 7
      status = mgrdimg(ri_id, start, stride, edges, subset_image)

C     Terminate access to the array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface and the file.
      status = mgend(gr_id)

C     Close the file.
      file_id = hclose(file_id)

      end
```

EXAMPLE 5.

Sampling Image Data

These examples illustrate how samples of data set elements can be read by using the `stride` argument of the **GRreadimage** function. Here we read every fourth row and every other column.

C:

```
#include "hdf.h"
#include "mfgr.h"

#define X_LENGTH 10
#define Y_LENGTH 20

main( )
{
int32 gr_id, ri_id, file_id, start[2], edges[2], ncomp, il;
int32 stride[2], dims[2];
int16 all_image[Y_LENGTH][X_LENGTH][2], sample_image[5][5][2], status;
intn i, j;

/* Open the file. */
file_id = Hopen("Example5.hdf", DFACC_CREATE, 0);

/* Open the file. */
gr_id = GRstart(file_id);

/* Define the number of components and dimensions of the image. */
ncomp = 2;
dims[0] = X_LENGTH;
dims[1] = Y_LENGTH;
il = MFGR_INTERLACE_PIXEL;

/*  Create the array. */
ri_id = GRcreate(gr_id, "Image_array_5", ncomp, DFNT_INT16, il, dims);

/*  Compute and store the data values. */
for (j = 0; j < Y_LENGTH; j++) {
    for (i = 0; i < X_LENGTH; i++) {
        all_image[j][i][0] = i + j * 10;
        all_image[j][i][1] = i + j * 10;
    }
}

/* Define the start and edge parameters. */
start[0] = start[1] = 0;
edges[0] = X_LENGTH;
edges[1] = Y_LENGTH;

/* Write the buffered data in all_image to the array. */
status = GRwriteimage(ri_id, start, NULL, edges, (VOIDP)all_image);

/* Close the GR interface and the file, then re-open both and
select the first and only data set in the file. */
status = GRendaccess(ri_id);
status = GRend(gr_id);
status = Hclose(file_id);
file_id = Hopen("Example5.hdf", DFACC_RDONLY, 0);
gr_id = GRstart(file_id);
ri_id = GRselect(gr_id, 0);

/* Read the data into sample_image, skipping every fourth \
row and every other column. */
```

```
start[0] = start[1] = 0;
edges[0] = 5;
edges[1] = 5;
stride[0] = 2;
stride[1] = 4;
status = GRreadimage(ri_id, start, stride, edges, (VOIDP)sample_image);

/* Terminate access to the array */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface the file */
status = GRend(gr_id);

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

}
```

FORTRAN:

```
      PROGRAM READ STRIDES

      integer*4 gr_id, ri_id, file_id, ncomp, il
      integer*4 start(2), edges(2), stride(2), dims(2)
      integer i, j, status
      integer mgstart, mgcreat, mgwrimg, mgendac
      integer mgselct, mgend, hopen, hclose

C     DFACC_CREATE and DFNT_INT16 are defined in hdf.h.
      integer*4 DFACC_CREATE, DFACC_RDONLY, DFNT_INT16
      integer*4 X_LENGTH, Y_LENGTH

      parameter (DFACC_CREATE = 4, DFACC_RDONLY = 1, DFNT_INT16 = 22,
     +           X_LENGTH = 10, Y_LENGTH = 20)
      integer*2 all_image(2, X_LENGTH, Y_LENGTH)

C     Create the file.
      file_id = hopen('Example5.hdf', DFACC_CREATE, 0)

C     Create the file.
      gr_id = mgstart(file_id)

C     Define the number of components and dimensions of the image.
      ncomp = 2
      dims(1) = X_LENGTH
      dims(2) = Y_LENGTH

C     Create the array.
      ri_id = mgcreat(gr_id, 'Image_array_5', ncomp, DFNT_INT16, il,
     +           dims)

C     Compute and store the data values.
      do 20 j = 1, Y_LENGTH
         do 10 i = 1, X_LENGTH
            all_image(1, i, j) = i + j * 10
            all_image(2, i, j) = i + j * 10
10       continue
20    continue

C     Set up the start and edge parameters to write the buffered
C     data to the entire array.
      start(1) = 0
      start(2) = 0
```

```
      edges(1) = X_LENGTH
      edges(2) = Y_LENGTH
      stride(1) = 1
      stride(2) = 1

C     Write the buffered data in all_image to the data set array.
      status = mgwrimg(ri_id, start, stride, edges, all_image)

C     Close the GR interface and the file, then re-open both and
C     select the first and only data set in the file.
      status = mgendac(ri_id)
      status = mgend(gr_id)
      status = hclose(file_id)
      file_id = hopen('Example5.hdf', DFACC_RDONLY, 0)
      gr_id = mgstart(file_id)
      ri_id = mgselct(gr_id, 0)

C     Read the data into sample_image, skipping every fourth row
C     and every other column.
      edges(1) = 5
      edges(2) = 5
      stride(1) = 4
      stride(2) = 2

C     Terminate access to the array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

      end
```

8.7 Obtaining Information About Files and General Raster Images

The routines covered in this section provide methods for obtaining information about the images in a file, for identifying images that meet certain criteria, and for obtaining information about the data sets themselves.

GRfileinfo gives the number of images and file attributes in a file, and **GRgetiminfo** provides information about individual images. To cycle through the images in a file, a calling program must use **GRfileinfo** to determine the number of images, followed by repeated calls to **GRgetiminfo** to view them. The parameters for these routines are described below. (See Table 8F on page 219.)

8.7.1 Obtaining Information about the Contents of a File: GRfileinfo

It is often useful to determine the number of images and global file attributes contained in a file before executing a series of read or write operations. The **GRfileinfo** routine is designed for this purpose. To determine the contents of a file, the calling program must contain the following:

C:

```
file_id = Hopen(filename, access_mode, number_of_desc);
gr_id = GRstart(file_id);
status = GRfileinfo(gr_id, &n_datasets, &n_file_attr);
status = GRend(gr_id);
status = Hclose(file_id);
```

FORTRAN:
```
file_id = hopen(filename, access_mode, number_of_desc)
gr_id = mgstart(filename, access_mode)
status = mgfinfo(gr_id, n_datasets, n_file_attr)
status = mgend(gr_id)
status = hclose(file_id)
```

The number of images in the file will be returned in the `n_datasets` argument, and `n_file_attr` will contain the total number of file attributes.

8.7.2 Obtaining Information About an Image: GRgetiminfo

It is impossible to allocate the proper amount of memory to buffer the image data when the number of components, dimension sizes and/or data type of the image are unknown. The **GRgetiminfo** routine provides necessary information about images for this purpose. To access information about an image, the calling program must contain the following:

C:
```
file_id = Hopen(filename, access_mode, number_of_desc);
gr_id = GRstart(file_id);
ri_id = GRselect(gr_id, gr_index);
status = GRgetiminfo(ri_id, name, &ncomp, &data_type, &il, dim_sizes,
         &n_attr);
status = GRend(gr_id);
status = GRendaccess(ri_id);
status = Hclose(file_id);
```

FORTRAN:
```
file_id = hopen(filename, access_mode, number_of_desc)
gr_id = mgstart(file_id)
ri_id = mgselct(gr_id, gr_index)
status = mggiinf(ri_id, name, ncomp, data_type, il, dim_sizes,
         n_attr)
status = mgend(gr_id)
status = mgendac(ri_id);
status = hclose(file_id)
```

GRgetiminfo takes a raster image identifier as input, and returns the name, number of components, data type, interlace mode, dimension size and number of attributes for the corresponding image in the `name`, `ncomp`, `nt`, `il`, `dim_sizes` and `n_attr` arguments respectively. The number of components of an image array element corresponds to the order of a vdata field, therefore this implementation of image components in the GR interface is flexible enough to accommodate any representation of pixel data. The calling program determines this representation; the GR interface recognizes only the raw byte configuration of the data. The attribute count will only reflect the number of attributes assigned to the image array; file attributes are not included.

TABLE 8F

GRfileinfo and GRgetiminfo Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRfileinfo (mgfinfo)	gr_id	int32	integer	General raster data set identifier.
	n_datasets	int32 *	integer	Number of data sets in the file.
	n_file_attr	int32 *	integer	Number of global attributes in the file.
GRgetiminfo (mggiinf)	ri_id	int32	integer	Image identifier.
	name	char *	character* (*)	Buffer for the name of the image.
	ncomp	int32 *	integer	Buffer for the number of components in the image.
	data_type	int32 *	integer	Buffer for the data type for the data in the image.
	il	int32 *	integer	Buffer for the interlace type of the data in the image.
	dim_sizes	int32 [2]	integer (2)	Buffer for the size of each dimension in the image.
	nattrs	int32 *	integer	Buffer for the number of attributes in the image.

EXAMPLE 6.

Retrieving Image Information

The following examples illustrate the use of the **GRgetiminfo** routine.

C:

```
#include "hdf.h"
#include "mfgr.h"

#define X_LENGTH 15
#define Y_LENGTH 10

main( )
{
int32 gr_id, ri_id, file_id, status, il;
int32 nt, dimsizes[2], ncomp, nattrs;
int32 start[2], edges[2];
int16 image_data[Y_LENGTH][X_LENGTH][2];
char name[MAX_GR_NAME];

/* Open the file. */
file_id = Hopen("Example2.hdf", DFACC_RDONLY, 0);

/* Initiate the GR interface. */
gr_id = GRstart(file_id);

/* Select the first (and in this case, only) image in the file. */
ri_id = GRselect(gr_id, 0);

/* Verify the characteristics of the image. */
status = GRgetiminfo(ri_id, name, &ncomp, &nt, &il, dimsizes, &nattrs);

/* Define the location, pattern, and size of the data to read from \
    the image. */
start[0] = start[1] = 0;
edges[0] = dimsizes[0];
edges[1] = dimsizes[1];

/* Read the array data set created in Example 2. */
status = GRreadimage(ri_id, start, NULL, edges, (VOIDP)image_data);

/* Terminate access to the image. */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface. */
status = GRend(gr_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM GET IMAGE INFO

      integer*4 gr_id, ri_id, file_id, il, ncomp
      integer*4 data_type, nattrs, status
      integer start(2), edge(2), stride(2)
      integer mgstart, mgselct, mggiinf
      integer mgrdimg, mgendac, mgend, hopen, hclose
      integer*4 DFACC_RDONLY, MAX_GR_NAME
```

```
      integer*4 X_LENGTH, Y_LENGTH
      parameter (DFACC_RDONLY = 1, MAX_GR_NAME = 256,
     +           X_LENGTH = 15, Y_LENGTH = 10)
      integer*2 image_data(2, X_LENGTH, Y_LENGTH)
      character name(MAX_GR_NAME)
      integer*4 dimsizes(2)

C     Open the file.
      file_id = hopen('Example2.hdf', DFACC_RDONLY, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Select the first (and in this case, only) image in the file.
      ri_id = mgselct(gr_id, 0)

C     Verify the characteristics of the image.
      status = mggiinf(ri_id, name, ncomp, data_type, il, dimsizes,
     +            nattrs)

C     Define the location, pattern, and size of the data to read
C     from the image.
      start(1) = 0
      start(2) = 0
      stride(1) = 1
      stride(2) = 1
      edge(1) = dimsizes(1)
      edge(2) = dimsizes(2)

C     Read the image.
      status = mgrdimg(ri_id, start, stride, edge, image_data)

C     Terminate access to the array data set.
      status = mgendac(ri_id)

C     Terminate access to the GR interface.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

      end
```

EXAMPLE 7.

Printing Image Names

The **GRgetiminfo** function can be called within a loop to retrieve the names of all images in an HDF file, as shown in the following examples.

C:

```
#include "hdf.h"

main( )
{
int32 gr_id, ri_id, file_id, n_datasets, n_file_attrs, gr_index;
int32 dim_sizes[2];
int32 ncomp, il, data_type, attributes, status;
char name[MAX_GR_NAME];

/* Open the file. */
```

```
file_id = Hopen("Example2.hdf", DFACC_RDONLY, 0);

/* Initiate the GR interface. */
gr_id = GRstart(file_id);

/* Determine the contents of the file. */
status = GRfileinfo(gr_id, &n_datasets, &n_file_attrs);

/* Access and print the name of each image in the file. */
for (gr_index = 0; gr_index < n_datasets; gr_index++) {
    ri_id = GRselect(gr_id, gr_index);
    status = GRgetiminfo(ri_id, name, &ncomp, &data_type, \
                &il, dim_sizes, &attributes);
    printf("name = %s\n", name);
    status = GRendaccess(ri_id);
}

/* Terminate access to the GR interface. */
status = GRend(gr_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM PRINT NAMES

      integer*4 gr_id, ri_id, file_id
      integer*4 n_datasets, n_file_attrs, index, status
      integer*4 attributes
      integer mgstart, mgfinfo, mgselct, mggiinf
      integer mgendac, mgend, hopen, hclose

      integer*4 DFACC_RDONLY, MAX_GR_NAME
      parameter (DFACC_RDONLY = 1, MAX_GR_NAME = 256)

      integer*4 dim_sizes(2)
      character name *(MAX_GR_NAME)

C     Open the file.
      file_id = hopen('Example2.hdf', DFACC_RDONLY, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Determine the contents of the file.
      status = mgfinfo(gr_id, n_datasets, n_file_attrs)

C     Access and print the names of every data set in the file.
      do 10 gr_index = 0, n_datasets - 1
         ri_id = mgselct(gr_id, index)
         status = mggiinf(ri_id, name, ncomp, data_type, il,
     +                    dim_sizes, attributes)
         print *, "name = ", name
         status = mgendac(ri_id)
10    continue

C     Terminate access to the GR interface.
      status = mgend(gr_id)

C     Close the file.
```

```
status = hclose(file_id)

end
```

8.7.3 Getting the Index of an Image: GRreftoindex and GRnametoindex

In addition to an index and name, images are also assigned a tag and reference number. **GRrefto-index** determines the index of the image from its reference number.

Selecting a data set from a given reference number involves the following steps:

1. Open the file.
2. Convert the reference number for the image into an index number.
3. Select the image by obtaining its identifier from its index number.

The calling program must contain the following:

C:
```
gr_id = GRstart(file_id);
gs_index = GRreftoindex(gr_id, ref);
ri_id = GRselect(gr_id, gs_index);
```

FORTRAN:
```
gr_id = mgstart(file_id)
gs_index = mgr2idx(gr_id, ref)
ri_id = mgselct(gr_id, gs_index)
```

GRreftoindex returns the index specified by the `gs_index` parameter for the image in the file with the reference number `ref`. The index `gs_index` can then be passed to **GRselect** to obtain an identifier for the image.

Images can also be selected using the image name as the search criteria. The calling sequence described above is the same for this purpose, except **GRnametoindex** is called instead of **GRref-toindex** and the image name is passed as the second parameter to **GRnametoindex**. As with **GRreftoindex**, **GRnametoindex** returns the index number of the target image.

TABLE 8G **GRreftoindex and GRnametoindex Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRreftoindex (mgr2idx)	gr_id	int32	integer	General raster data set identifier.
	ref	uint16	integer	Reference number of the target image.
GRnametoindex (mgn2ndx)	gr_id	int32	integer	General raster data set identifier.
	name	char *	character (*)	Name of the target image.

EXAMPLE 8. **Searching for an Image Index**

If the index of a image isn't known, the **GRnametoindex** function can be used to retrieve the index based on a given data set name. In these examples, an invalid image name is given which results in **GRnametoindex** returning a value of `-1`. A valid image name is then provided and the returned index is used to read the contents of the corresponding image.

C:

```
#include "hdf.h"

#define X_LENGTH 10
```

```
#define Y_LENGTH 20

main( )
{

int32 gr_id, gr_index, ri_id, file_id, status;
int32 start[2], edges[2];
int16 image_data[Y_LENGTH][X_LENGTH][2];

/* Open the file. */
file_id = Hopen("Example5.hdf", DFACC_RDONLY, 0);

/* Initiate the GR interface. */
gr_id = GRstart(file_id);

/* Search for the index of a non-existent image. */
gr_index = GRnametoindex(gr_id, "Invalid_Data_Set_Name");

/* Error condition: gr_index contains the value -1. */

/* Search for the index of the image named "Image_array_5". */
gr_index = GRnametoindex(gr_id, "Image_array_5");

/* Select the image corresponding to the returned index. */
ri_id = GRselect(gr_id, gr_index);

/* Read the data set data into the array_data array. */
start[0] = start[1] = 0;
edges[0] = X_LENGTH;
edges[1] = Y_LENGTH;
status = GRreadimage(ri_id, start, NULL, edges, (VOIDP)image_data);

/* Terminate access to the array */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface. */
status = GRend(gr_id);

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

}
```

FORTRAN:

```
     PROGRAM SEARCH INDEX

      integer*4 gr_id, gr_index, ri_id, file_id, status
      integer*4 start(2), edges(2), stride(2)
      integer mgstart, mgn2ndx, mgrdimg, mgendac, mgselct
      integer mgend, hopen, hclose

C     DFACC_RDONLY is defined in hdf.h.
      integer*4 DFACC_RDONLY
      integer*4 X_LENGTH, Y_LENGTH
      parameter (DFACC_RDONLY = 1, X_LENGTH = 10, Y_LENGTH = 20)
      integer*2 image_data(2, X_LENGTH, Y_LENGTH)

C     Open the file.
      file_id = hopen('Example5.hdf', DFACC_RDONLY, 0)

C     Initialize the GR interface.
      gr_id = mgstart(file_id)
```

```
C     Search for the index of a non-existent image.
      gr_index = mgn2ndx(gr_id, 'Invalid_Data_Set_Name')

C     Error condition: gr_index contains the value -1.
C     Search for the index of the image named 'Image_array_5'.
      gr_index = mgn2ndx(gr_id, 'Image_array_5')

C     Select the image corresponding to the returned index.
      ri_id = mgselct(gr_id, gr_index)

C     Read the image data into the image_data array.
      start(1) = 0
      start(2) = 0
      edges(1) = X_LENGTH
      edges(2) = Y_LENGTH
      stride(1) = 1
      stride(2) = 1
      status = mgrdimg(ri_id, start, stride, edges, image_data)

C     Terminate access to the array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface and the file.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

      end
```

8.7.4 Compressing General Raster Images

Images can be compressed by calling the **GRsetcompress** routine. **GRsetcompress** compresses the image data at the time it is called and supports all standard HDF compression algorithms. The syntax of the **GRsetcompress** routine is as follows:

C: `status = GRsetcompress(ri_id, comp_type, c_info);`

As with **SDsetcompress**, the `comp_type` argument is the compression type definition and all compression method definitions supported in **SDsetcompress** are also supported in **GRsetcompress**. The `c_info` argument is a structure of type `comp_info` - this structure is defined in the "hcomp.h" header file and in Chapter 3 titled *Scientific Data Sets (SD API)*. It contains the algorithm-specific information necessary for **GRsetcompress** to perform the specified data compression.

TABLE 8H **GRsetcompress Parameter List**

Routine Name	Parameter	Data Type	Description
		C	
GRsetcompress	ri_id	int32	Image identifier.
	comp_type	int32	Compression method.
	c_info	comp_info *	Pointer to compresion infomation structure.

8.7.5 External File Operations Using the GR API

An ***external image array*** is one that is stored in a file that is not the file containing the metadata for the image. The concept of externally stored data is described in Chapter 3, titled *Scientific Data Sets (SD API)*. The GR interface supports the same functionality as the SD interface.

8.7.5.1 Creating a General Raster Image in an External File

Creating an image using external data involves the same general steps as with the SD interface:

1. Create the image array.
2. Specify that an external data file is to be used.
3. Write data to the image array.
4. Terminate access to the image.

To create a data set containing an external file, the calling program must make the following calls.

C:
```
ri_id = GRcreate(gr_id, name, ncomp, nt, il, dim_sizes);
status = GRsetexternalfile(ri_id, filename, offset);
status = GRwriteimage(ri_id, start, stride, edge, image_data);
status = GRendaccess(ri_id);
```

FORTRAN:
```
ri_id = mgcreat(gr_id, name, ncomp, nt, il, dim_sizes)
status = mgsxfil(ri_id, filename, offset)
status = mgwrimg(ri_id, start, stride, edge, image_data)
status = mgendac(ri_id)
```

GRsetexternalfile marks the image identified by `ri_id` as one whose data is to be written to an external file. The parameter `filename` is the name of the external data file, and `offset` is the number of bytes from the beginning of the external file to the location where the first byte of data will be written.

GRsetexternalfile can only be called once per data set. If a file with the same name as `filename` exists in the current directory, HDF will use it as the external file. If the file does not exist, HDF will create one. Once the external filename is specified, it is impossible to change it without breaking the association between the general raster data set and its image data.

Use caution when writing to existing files because the **GRwriteimage** library begins its write at the specified offset without checking whether data is being overwritten. When different data sets have arrays occupying the same external file, the calling program is responsible for avoiding any overlap between them.

For more information on the parameters used in **GRsetexternalfile** refer to the following table.

TABLE 8I **GRsetexternalfile Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRsetexternalfile (**mgsxfil**)	ri_id	int32	integer	Image identifier.
	filename	char *	character* (*)	Name of the external data set.
	offset	int32	integer	Offset in bytes from the beginning of the external file to the image data

8.7.5.2 Moving General Raster Images to an External File

Images can be moved from a primary file to an external file. To do so requires the following steps:

1. Select the image.
2. Specify the external data file.
3. Terminate access to the image.

The calling program must make the following calls:

C:
```
ri_id = GRselect(gr_id, gr_index);
status = GRsetexternalfile(ri_id, filename, offset);
status = GRendaccess(ri_id);
```

When **GRsetexternalfile** is used in conjunction with **GRselect**, it will immediately write the existing data to the external file and any data in the external file that occupies the space reserved for the external array will be overwritten as a result of this operation. A data set can only be moved to an external file once.

During the operation, the data is written to the external file as a contiguous stream regardless of how it is stored in the primary file. Because data is moved "as is", any unwritten locations in the data set are preserved in the external file. Subsequent read and write operations performed on the data set will access the external file.

8.7.5.3 Setting the I/O Mode for External General Raster Image Access

In situations where it is necessary to perform parallel I/O access to external image objects - for instance, if the external image data resides on a RAID array and is to be accessed by a multiprocessor system - the **GRsetaccesstype** routine can be used to set the access mode to either parallel or serial I/O. The syntax of the **GRsetaccesstype** routine is as follows:

C:
```
status = GRsetaccesstype(ri_id, access_mode);
```

FORTRAN:
```
status = mgsactp(ri_id, access_mode)
```

The `access_mode` argument can be set to either `DFACC_SERIAL` for serial mode or `DFACC_PARALLEL` for parallel mode. **GRsetaccesstype** returns a status code of `0` for successful completion or `1` if an error occurred.

TABLE 8J **GRsetaccesstype Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRsetaccesstype (**mgsactp**)	ri_id	int32	integer	Image identifier.
	access_mode	char *	character* (*)	Access mode.

8.8 General Raster Image Attributes

The concepts of user-defined and predefined attributes are explained in Chapter 3, titled *Scientific Data Sets (SD API)*. The GR implementation of attributes is similar to the SD implementation. Attributes are not written out to a file until access to the object the attribute is attached to is terminated.

8.8.1 Predefined GR Attributes

The GR interface library has only one predefined attribute: `FILL_ATTR`. This attribute defines a fill pixel, which is analogous to a fill value in the SD interface. It represents the default value that is written to each element of an image array not explicitly written to by the calling program; when only a portion of the entire image array is filled with data. This value must of the same data type as the rest of the initialized image data. The routine used to set the fill value is **GRsetattr**, explained in the next section.

8.8.2 Setting User-Defined Attributes: GRsetattr

Setting an attribute involves the following steps:

1. Open the file.
2. Initialize the GR interface.
3. Set the attribute.
4. Terminate access by disposing of any existing identifiers.
5. Terminate access to the GR interface.
6. Close the file.

To assign an attribute to a file, the calling program must contain the following sequence calls:

C:

```
gr_id = GRstart(filename, access_mode);
status = GRsetattr(gr_id|ri_id, attr_name, data_type, count,
        attr_value);
status = GRend(gr_id);
```

FORTRAN:

```
gr_id = mgstart(filename, access_mode)
status = mgsnatt(gr_id|ri_id, attr_name, data_type, count,
        attr_value)
status = mgend(gr_id)
```

In **GRsetattr** the first argument can either be the id of a file, palette or raster image. The attribute will be assigned to the object referred to by this identifier. The argument `attr_name` contains the name of the attribute and can be no more than `MAX_GR_NAME` characters in length. Passing the name of an existing attribute will overwrite the value portion of that attribute.

The arguments, `data_type`, `count` and `attr_value` describe the right side of the `label=value` equation. The `attr_value` argument contains one or more values of the same data type. The `data_type` argument describes the data type for all values in the attribute and `count` contains the total number of values in the attribute.

As with other HDF routines that accommodate numeric and character data, the **GRsetattr** routine has two Fortran-77 versions: **mgsnatt** and **mgscatt**. The **mgsnatt** routine writes numeric attribute data and **mgscatt** writes character attribute data.

The parameters for **GRsetattr** are further described below. (See Table 8K on page 231.) Note that, because there are two Fortran-77 versions of **GRsetattr**, there are correspondingly two entries in the "Data Type" field of the *values* parameter.

EXAMPLE 9. **Setting File and Image Attribute Values**

The following code segment calls **GRsetattr** to assign file and image attributes to a general raster data set.

C:

```
#include "hdf.h"

main( )
```

```
{
int32 gr_id, ri_id, file_id, dim_index, status;
int32 num_values[2];

/* Open the file. */
file_id = Hopen("Example5.hdf", DFACC_RDWR, 0);

/* Initialize the GR interface. */
gr_id = GRstart(file_id);

/* Set an attribute that describes the file contents. */
status = GRsetattr(gr_id, "File contents", DFNT_CHAR8, 16, \
                (VOIDP)"RGB image data");

/* Get the identifier for the first data set. */
ri_id = GRselect(gr_id, 0);

/* Set an attribute the specifies a valid range of values. */
num_values[0] = 2;
num_values[1] = 10;
status = GRsetattr(ri_id, "Value range", DFNT_INT32, 2, \
                (VOIDP)num_values);

/* Terminate access to the image. */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface. */
status = GRend(gr_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM SET IMAGE ATTRIBS

      integer*4 gr_id, ri_id, file_id, status
      integer num_values(2)
      integer mgstart, mgsnatt, mgselct
      integer mgendac, mgend, hopen, hclose
      integer DFACC_RDWR, DFNT_CHAR8, DFNT_INT32
      parameter (DFACC_RDWR = 3, DFNT_CHAR8 = 4, DFNT_INT32 = 24)

C     Open the file.
      file_id = hopen('Example5.hdf', DFACC_RDWR, 0)

C     Initialize the GR interface.
      gr_id = mgstart(file_id)

C     Set an attribute that describes the file contents.
      status = mgsnatt(gr_id, 'File contents', DFNT_CHAR8, 16,
     +                 'RGB image data')

C     Get the identifier for the first data set.
      ri_id = mgselct(gr_id, 0)

C     Set an attribute the specifies a valid range of values.
      num_values(1) = 2
      num_values(2) = 10
      status = mgsnatt(ri_id, 'Value range', DFNT_INT32, 2, num_values)
```

```
C     Terminate access to the image.
      status = mgendac(ri_id)

C     Terminate access to the GR interface.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

      end
```

8.8.3 Querying User-Defined Attributes: GRfindattr and GRattrinfo

Each attribute associated with an object has a unique ***attribute index***, a value ranging from 0 to the total number of attributes in the object. Given a file, array or dimension id and an attribute name, **GRfindattr** will return a valid attribute index if the attribute exists. The attribute index can then be used to retrieve information about an attribute or its values. Given a file or array and a valid attribute index **GRattrinfo** returns the name, data type and count for the attribute if it exists.

The syntax for **GRfindattr** and **GRattrinfo** is as follows:

C:
```
status = GRfindattr(ri_id|gr_id, attr_name);
status = GRattrinfo(ri_id|gr_id, attr_index, attr_name,
        &data_type, &count);
```

FORTRAN:
```
status = mgfndat(ri_id|gr_id, attr_name)
status = mgatinf(ri_id|gr_id, attr_index, attr_name,
        data_type, count);
```

The parameters for **GRfindattr** and **GRattrinfo** are further described below. (See Table 8K.)

8.8.4 Reading User-Defined Attributes: GRgetattr

GRgetattr reads an attribute's values. The syntax for **GRgetattr** is as follows:

C:
```
status = GRgetattr(ri_id|gr_id, attr_index, data);
```

FORTRAN:
```
status = mggnatt(ri_id|gr_id, attr_index, data)
```

GRgetattr takes a file, array, or dimension identifier and an attribute index specified in the `attr_index` parameter as input parameters and returns the attribute's values in the buffer `data`.

It's assumed that the buffer `data`, allocated to hold the attribute values, is large enough to hold the data - if not, the data read will be truncated to the size of the buffer. The size of the buffer should be at least `count*DFKNTsize(number_type)` bytes long. If an attribute contains multiple values, **GRgetattr** will return all of them. It is not possible to read a subset of values.

As with **GRsetattr**, **GRgetattr** has two Fortran-77 versions: **mggnatt** and **mggcatt**. The **mggnatt** routine reads numeric attribute data and **mggcatt** reads character attribute data.

The parameters for **GRgetattr** are further described in Table 8K. Note that, because there are two Fortran-77 versions of **GRgetattr**, there are correspondingly two entries in the "Data Type" field of the *values* parameter.

TABLE 8K

GRsetattr, GRfindattr, GRattrinfo and GRgetattr Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRsetattr (mgsnatt/ mgscatt)	*_id	int32	integer	GR data set or image identifier.
	attr_name	char *	character* (*)	Name assigned to the attribute.
	data_type	int32	integer	Specifies the data type.
	count	int32	integer	Number of values in the attribute.
	values	VOIDP	<valid numeric data type>	Buffer of the data to be written.
GRfindattr (mgfadat)	*_id	int32	integer	GR data set or image identifier.
	attr_name	char *	character* (*)	Name assigned to the attribute.
GRattrinfo (mgatinf)	*_id	int32	integer	GR data set or image identifier.
	attr_index	int32	integer	Index for the attribute to be read.
	attr_name	char *	character* (*)	Buffer for the name of the dimension attribute.
	data_type	int32 *	integer	Buffer for the data type of the values in the attribute.
	count	int32 *	integer	Buffer for the total number of values in the attribute.
GRgetattr (mggnatt/ mggcatt)	*_id	int32	integer	GR data set or image identifier.
	attr_index	int32	integer	Index for the attribute to be read.
	values	VOIDP	<valid numeric data type>	Buffer for the attribute values.

EXAMPLE 10.

Retrieving Image Attribute Information

The image attribute information stored in the "Example4.hdf" HDF file in Example 4 are read from the file in these examples.

C:

```
#include “hdf.h”

main( )
{

int32 gr_id, ri_id, file_id, status;
int32 attr_index, data_type, count;
char attr_name[MAX_GR_NAME];
int8 *buffer;

/* Open the file. */
file_id = Hopen(“Example5.hdf”, DFACC_RDONLY, 0);

/* Initialize the GR interface. */
gr_id = GRstart(file_id);

/* Find the file attribute named “File contents”. */
attr_index = GRfindattr(gr_id, “File contents”);

/* Get information about the file attribute. */
status = GRattrinfo(gr_id, attr_index, attr_name, &data_type, &count);

/* Allocate a buffer to hold the attribute data. */
buffer = HDmalloc(count * DFKNTsize(data_type));

/* Read the attribute data. */
status = GRgetattr(gr_id, attr_index, buffer);
```

```
/* Free the buffer memory. */
HDfree(buffer);

/* Get the identifier for the first image. */
ri_id = GRselect(gr_id, 0);

/* Find the data set attribute named "Value range". */
attr_index = GRfindattr(ri_id, "Value range");

/* Get information about the image attribute. */
status = GRattrinfo(ri_id, attr_index, attr_name, &data_type, &count);

/* Allocate a buffer to hold the attribute data. */
buffer = HDmalloc(count * DFKNTsize(data_type));

/* Read the attribute data. */
status = GRgetattr(ri_id, attr_index, buffer);

/* Terminate access to the array */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface. */
status = GRend(gr_id);

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

}
```

FORTRAN:

```
      PROGRAM IMAGE ATTRIB INFO

       integer*4 gr_id, ri_id, file_id
       integer attr_index, data_type, count, status
       character attr_name *13
       character buffer *20

       integer mgstart, mgfndat, mgatinf, mggnatt, mgselct
       integer mgendac, mgend, hopen, hclose

C      DFACC_RDONLY is defined in hdf.h.
       integer DFACC_RDONLY
       parameter (DFACC_RDONLY = 1)

C      Open the file.
       file_id = hopen('Example5.hdf', DFACC_RDONLY, 0)

C      Open the file.
       gr_id = mgstart(file_id)

C      Find the file attribute named 'File contents'.
       attr_index = mgfndat(gr_id, 'File contents')

C      Get information about the file attribute.
       status = mgatinf(gr_id, attr_index, attr_name, data_type, count)

C      Read the attribute data.
       status = mggnatt(gr_id, attr_index, buffer)

C      Get the identifier for the first image.
       ri_id = mgselct(gr_id, 0)
```

```
C     Find the data set attribute named "Value range".
      attr_index = mgfndat(ri_id, 'Value range')

C     Get information about the image attribute.
      status = mgatinf(ri_id, attr_index, attr_name, data_type, count)

C     Read the attribute data.
      status = mggnatt(ri_id, attr_index, buffer)

C     Terminate access to the array.
      status = mgendac(ri_id)

C     Terminate access to the GR interface.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)
      end
```

8.9 Reading and Writing Palette Data Using the GR API

The GR API library includes routines that read, write and access information about palette data attached to general raster images. Although this functionality is also provided by the HDF Palette API library it is not a recommended practice to use the Palette API to access and manipulate palette objects created by GR API routines.

The routines are named **GRgetlutid**, **GRgetlutinfo**, **GRluttoref**, **GRwritelut** and **GRreadlut**. Note that the routine names use the term ***LUT*** to refer to palettes, which stands for color ***lookup tables***.

8.9.1 Obtaining a Palette ID: GRgetlutid

The **GRgetlutid** function takes two arguments, the raster image identifier of the image that has the target palette attached to it and the index of the target palette, and returns the palette identifier corresponding to the specified image as its return value.

The syntax of the **GRgetlutid** function is:

C: `pal_id = GRgetlutid(ri_id, lut_index);`

FORTRAN: `pal_id = mggltid(ri_id, lut_index)`

8.9.2 Obtaining Palette Information: GRgetlutinfo

The **GRgetlutinfo** function takes one input argument, the identifier of the palette, and returns the number of components of the palette, the type of data contained in the palette, the interlace mode of the stored palette data and the number of entries in the palette in the `ncomp`, `data_type`, `interlace_mode` and `num_entries` arguments respectively. It returns a status code of `0` on successful completion and `-1` if an error occurred.

The syntax of the **GRgetlutinfo** function is:

C:
```
status = GRgetlutinfo(pal_id, &ncomp, &data_type, &interlace_mode,
         num_entries);
```

FORTRAN: `status = mgglinf(pal_id, ncomp, data_type, interlace_mode, num_entries)`

8.9.3 Retrieving the Reference Number of the Specified Palette: GRluttoref

The **GRluttoref** routine takes one argument, a palette identifier, and returns the reference number of the palette. **GRluttoref** is commonly used to annotate the palette or including the palette within a vgroup

There is currently no Fortran-77 version of the **GRluttoref**.

TABLE 8L **GRluttoref Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRluttoref	pal_id	int32	None	Image identifier.

8.9.4 Writing Palette Data: GRwritelut

The **GRwritelut** function takes six input arguments, the identifier of the palette, the number of components of the palette, the type of data contained in the palette, the interlace mode of the stored palette data, the number of entries in the palette and the buffered palette data to be written in the `pal_id`, `ncomp`, `data_type`, `interlace_mode`, `num_entries` and `pal_data` arguments respectively. It writes the palette data into the palette that is attached to the image specified by the given image identifier and returns a status code of `0` on successful completion and `-1` if an error occurred.

The syntax of the **GRwritelut** function is:

C: `status = GRwritelut(pal_id, ncomp, data_type, interlace_mode, num_entries, pal_data);`

FORTRAN: `status = mgwrlut(pal_id, ncomp, data_type, interlace_mode, num_entries, pal_data)`

There are two Fortran-77 versions of **GRwritelut**: **mgwrlut** and **mgwclut**. The **mgwrlut** routine writes buffered numeric palette data and **mgwclut** writes buffered character palette data.

8.9.5 Reading Palette Data: GRreadlut

The **GRreadlut** function takes two arguments, a palette identifier and a buffer large enough to store the read palette data. It reads the data into the buffer pointed to by the `pal_data` argument and returns a status code of `0` on successful completion and `-1` if an error occurred. The palette data is read according to the interlacing mode set by the last call to **GRreqlutil**.

The syntax of the **GRreadlut** function is:

C: `status = GRreadlut(pal_id, pal_data);`

FORTRAN: `status = mgrdlut(pal_id, pal_data)`

There are two Fortran-77 versions of **GRreadlut**: **mgrdlut** and **mgrclut**. The **mgrdlut** routine reads numeric palette data and **mgrclut** reads character palette data.

TABLE 8M

GRgetlutid, GRgetlutinfo, GRwritelut and GRreadlut Parameter List

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRgetlutinfo (mgglinf)	pal_id	int32	integer	Palette identifier.
	ncomp	int32*	integer (*)	Buffer for the number of components in each palette element.
	data_type	int32*	integer (*)	Buffer for the data type of the palette data.
	interlace	int32*	integer (*)	Buffer for the interlace type of the palette data.
	num_entries	int32*	integer (*)	Buffer for the size of the palette.
GRwritelut (mgwrlut/ mgwclut)	pal_id	int32	integer	Palette identifier.
	ncomp	int32	integer	Number of components in each palette element.
	data_type	int32	integer	Type of the palette data.
	interlace	int32	integer	Interlace type of the palette data.
	num_entries	int32	integer	Number of entries in the palette.
	pal_data	VOIDP	<valid numeric data type>	Palette data to be written.
GRgetlutid (mggltid)	ri_id	int32	integer	General raster image identifier.
	index	int32	integer	Image index.
GRreadlut (mgrdlut/ mgrclut)	pal_id	int32	integer	Palette identifier.
	pal_data	VOIDP	<valid numeric data type>	Buffer for the palette data to be read.

EXAMPLE 11.

Reading and Writing a Palette and Obtaining Palette Information

In the following code examples, **GRgetlutid, GRgetlutinfo, GRwritelut and GRreadlut** are used to write a palette to an HDF file named "Example12.hdf", then read it into a buffer and retrieve information about it.

C:

```
#include “hdf.h”
#include “mfgr.h”

#define X_LENGTH 20
#define Y_LENGTH 20

main( )
{
int32 gr_id, ri_id, file_id, pal_id, status, num_entries;
int32 data_type, ncomp, num_comp, interlace_mode;
int32 r_num_entries, r_data_type, r_ncomp, r_interlace_mode;
uint8 palette_data[256*3];
uint8 r_palette_data[256*3];
intn i, j;
int32 dims[2];

/* Create and open the file. */
file_id = Hopen(“Example11.hdf”, DFACC_CREATE, 0);

/* Initiate the GR interface. */
gr_id = GRstart(file_id);
```

```
/* Define the number of components and dimensions of the image. */
ncomp = 1;
dims[0] = X_LENGTH;
dims[1] = Y_LENGTH;
interlace_mode = MFGR_INTERLACE_PIXEL;

/* Create the image. */
ri_id = GRcreate(gr_id, "Image_12", ncomp, DFNT_UINT8,
                interlace_mode, dims);

/* 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;
}

/* Set palette characteristics. */
data_type = DFNT_UINT8;
num_entries = 256;
num_comp = 3;

/* Get the id for the palette. */
pal_id = GRgetlutid(ri_id, 0);

/* Write the palette to file. */
status = GRwritelut(pal_id, num_comp, data_type,
                interlace_mode, num_entries,
                (VOIDP)palette_data);

/* Read the palette information. */
status = GRgetlutinfo(pal_id, &r_ncomp, &r_data_type,
                &r_interlace_mode, &r_num_entries);

/* Read the palette data. */
status = GRreadlut(pal_id, (VOIDP)r_palette_data);

/* Terminate access to the image. */
status = GRendaccess(ri_id);

/* Terminate access to the GR interface. */
status = GRend(gr_id);

/* Close the file. */
status = Hclose(file_id);

}
```

FORTRAN:

```
     PROGRAM READ WRITE PALETTE

      integer*4 gr_id, ri_id, file_id, pal_id, status
      integer*4 num_entries, data_type, ncomp, interlace_mode
      integer*4 r_interlace_mode, r_num_entries, r_data_type
      integer*4 r_ncomp, ncomp, num_comp, dims(2)
      character palette_data(256*3)
      character r_palette_data(256*3)
      integer i
      integer*4 MFGR_INTERLACE_PIXEL, DFNT_CHAR8, X_LENGTH
      integer*4 Y_LENGTH
      parameter(MFGR_INTERLACE_PIXEL = 0, DFNT_CHAR8 = 4,
```

```
     +          X_LENGTH = 20, Y_LENGTH = 20)
      integer hopen, hclose, mgstart, mgcreat, mgwrlut
      integer mggltid, mgglinf, mgrdlut, mgendac, mgend

C     Create and open the file.
      file_id = hopen('Example11.hdf', DFACC_CREATE, 0)

C     Initiate the GR interface.
      gr_id = mgstart(file_id)

C     Define the Number of components and dimensions of the image
C     and palette to be created.
      ncomp = 1
      num_entries = 256
      num_comp = 3
      dims(1) = X_LENGTH
      dims(2) = Y_LENGTH
      interlace_mode = MFGR_INTERLACE_PIXEL
      data_type = DFNT_CHAR8

C     Create the image.
      ri_id = mgcreat(gr_id, 'Image_12', ncomp, data_type,
     +                interlace_mode, dims)

C     Initialize the palette to greyscale.
      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     Get the identifier for the palette.
      pal_id = mggltid(ri_id, 0)

C     Write the palette to the HDF file.
      status = mgwrlut(pal_id, num_comp, data_type,
     +                 interlace_mode, num_entries,
     +                 palette_data)

C     Read the palette information.
      status = mgglinf(pal_id, ncomp, data_type,
     +                 interlace_mode, num_entries)

C     Read the palette data.
      status = mgrdlut(pal_id, r_ncomp, r_data_type,
     +                 r_interlace_mode, r_num_entries,
     +                 r_palette_data)

C     Terminate access to the image.
      status = mgendac(ri_id)

C     Terminate access to the GR interface.
      status = mgend(gr_id)

C     Close the file.
      status = hclose(file_id)

      end
```

8.10 GR Object Identifier Conversion Functions

There are situations where it is necessary to convert, or map, one type of object identifier to another. One instance is when an image is to be attached to a vgroup, another is when an image's index must be obtained so that it can be passed to the **GRselect** routine to access the image, but all that is available is the image's reference number. The former instance requires a means of mapping a raster image identifier to a reference number, which in turn will be stored in a vgroup object. The latter instance requires a means of mapping an image's reference number to an index. The **GRidtoref** and **GRreftoindex** functions provide this functionality.

8.10.1 Mapping an Image Identifier to a Reference Number: GRidtoref

The **GRidtoref** routine takes one argument, an image identifier, and maps it to a valid reference number.

8.10.2 Mapping a Reference Number to an Index: GRreftoindex

The **GRreftoindex** routine takes two arguments, a file identifier and the reference number of an image, and maps them to an image index.

TABLE 8N **GRidtoref and GRreftoindex Parameter List**

Routine Name (Fortran-77)	Parameter	Data Type		Description
		C	Fortran-77	
GRidtoref (mgid2rf)	ri_id	int32	integer	Image identifier.
GRreftoindex (mgr2idx)	gr_id	int32	integer	General raster data set identifier.
	ref	uint16	integer	Reference number to be mapped.