* General Raster Images (GR API)

# Chapter Overview

This chapter describes the general raster (GR) data model, the GR interface (also called the GR API), and the interface routines used to manipulate GR data objects. The GR data model is designed to provide a flexible means of manipulating raster images. There were two other interfaces that worked with raster images, the DFR8 interface (Chapter 6, 8-Bit Raster Images (DFR8 API)) and the DF24 interface (Chapter 7, 24-bit Raster Images (DF24 API)) but the GR interface supersedes them.

## The GR Data Model

HDF users familiar with the SD interface will find certain aspects of the GR data model similar to the SD data model. The interfaces are similar in that both interfaces support data storage in multiple files, attributes, compression, and chunking. They are dissimilar in that palettes can be created and attached to an image through GR interface routines, customized dimension information is not supported in the GR interface, and GR dataset chunking is constrained to two dimensions.

* GR Data Set Contents

The terms GR data set, raster image, and image are used interchangeably in this chapter.



Refer to Figure 8a on page 295 for a graphical overview of the raster image, or GR data set, structure. Note that GR data sets consist of required and optional components.

### Required GR Data Set Components

Every GR data set must contain the following components: image array, name, pixel type, and dimensions. The name, dimensions, and pixel type must be supplied by the user at the time the GR data set is defined.

Image Array

An image array is a two-dimensional array of pixels. This is the primary data component of the GR model and will be discussed later in this section; it can be compressed, chunked, and/or stored in external files. Refer to Section 8.6.2 on page 312 for a description of raster image compression and Section 8.6.4 on page 313 for a description of external image storage.

A raster image has an index and a reference number associated with it. The index is a non-negative integer that describes the relative position of the raster image in the file. A valid index ranges from 0 to the total number of images in the file minus 1. The reference number is a unique positive integer assigned to the raster image by the GR interface when the image is created. Various GR interface routines can be used to obtain an image index or reference number depending on the available information about the raster image. The index can also be determined if the sequence in which the images are created in the file is known.

In the GR interface, a raster image identifier uniquely identifies a raster image within the file. The identifier is generated by the GR interface access routines when a new GR data set is created or an existing one is selected. The identifier is then used by other GR interface routines to access the raster image until the access to this image is terminated. For an existing raster image, the index of the image can be used to obtain the identifier.

Image Array Name

Each image array has a name consisting of a string of case-sensitive alphanumeric characters. The name must be provided by the calling program at the time the image is created, and cannot be changed afterward. Image array names do not have to be unique within a file, but if they are not it can be difficult to distinguish among the raster images in the file.

Pixels and Pixel Type

Each element in an image array corresponds to one pixel and each pixel can consist of a number of color component values or pixel components, e.g., Red-Green-Blue or RGB, Cyan-Magenta-Yellow-Black or CMYK, etc. Pixel components can be represented by different methods (8-bit lookup table or 24-bit direct representation, graphically depicted by Figure 6a on page 261 and Figure 7b on page 280, respectively) and may have different data types.

The data type of pixel components and the number of components in each pixel are collectively known as the pixel type. The GR data model supports all of the HDF-supported data types. A list of these data types appears provided in (See Table 2F on page 14).

Pixels can be composed of any number of components.

Dimensions

Image array dimensions specify the shape of the image array. A raster image array has two limited dimensions. The size of each dimension must be specified at the creation of the image and must be greater than 0.

The GR library does not allow the HDF user to add attributes to a dimension or to set dimension scale.

### Optional GR Data Set Components

There are two types of optional components available for inclusion in a GR data set: ***palettes*** and ***attributes***. These components are only created when specifically requested by the calling program; the GR interface does not provide predefined palettes or attributes.

Palettes

Palettes are lookup tables attached to images and define a set of color values for each pixel value in the image array. The GR interface provides similar capabilities for storing and manipulating palettes as the DFP interface described in Chapter 9, 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. In the meantime, the single-file DFP routines are fully compatible with palettes created with the GR palette routines. The GR palette routines are described in Section 8.11 on page 338.

Attributes

Attributes contain auxiliary information about a file, a raster image, or both. The concept of attributes is described in Chapter 3, Scientific Data Sets (SD API).

The GR interface does not support dimension attributes.

## The GR Interface

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

### GR Interface Routines

All C routine names in the GR 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 raster images.
* Raster image manipulation routines modify the data and metadata contained in a GR data set.
* LUT manipulation routines modify the palettes, also called color lookup tables or LUTs, contained in a GR data set.
* Maintenance routines create the data and metadata contained in a GR data set and modify global settings governing the format of the stored data.
* Inquiry routines return information about data contained in a GR data set.
* Chunking routines are used to define data chunking parameters, to retrieve chunking information, and to write and read chunked GR data sets.

The GR routines are listed in the following table and described further in subsequent sections of this chapter.

* GR Library Routines

|  |  |  |  |
| --- | --- | --- | --- |
| Purpose | Routine Name7 | | Description |
| C | FORTRAN-77 |
| Access | GRstart | mgstart | Initializes the GR interface (Section 8.5.1 on page 300) |
| GRcreate | mgcreat | Creates a new raster image (Section 8.5.1 on page 300) |
| GRselect | mgselct | Selects the raster image (Section 8.5.1 on page 300) |
| GRendaccess | mgendac | Terminates access to the raster image (Section 8.5.2 on page 301) |
| GRend | mgend | Terminates access to the GR interface (Section 8.5.2 on page 301) |
| Raster Image Manipulation | GRgetattr | mggnatt/  mggcatt | Reads an attribute of a raster image or a file (Section 8.10.4 on page 333) |
| GRidtoref | mgid2rf | Maps a raster image identifier to a reference number (Section 8.9.3 on page 322) |
| GRnametoindex | mgn2ndx | Maps the name of a raster image name to an index (Section 8.9.5 on page 323) |
| GRreadimage | mgrdimg/  mgrcimg | Reads raster image data (Section 8.7.1 on page 315) |
| GRreftoindex | mgr2idx | Maps the reference number of a raster image to its index (Section 8.9.4 on page 323) |
| GRsetattr | mgsnatt/  mgscatt | Assigns an attribute to a raster image or a file (Section 8.10.2 on page 329) |
| GRwriteimage | mgwrimg/mgwcimg | Writes raster image data (Section 8.6.1 on page 302) |
| GRreqimageil | mgrimil | Sets the interlace mode of the image read for subsequent read operations (Section 8.7.2 on page 315) |
| LUT Manipulation | GRgetlutid | mggltid | Gets a palette identifier given the palette’s index (Section 8.11.1 on page 338) |
| GRluttoref | mglt2rf | Maps a palette identifier to a reference number (Section 8.11.3 on page 339) |
| GRreadlut | mgrdlut/  mgrclut | Reads palette data from a raster image (Section 8.11.7 on page 341) |
| GRwritelut | mgwrlut/  mgwclut | Writes palette data to a raster image (Section 8.11.5 on page 340) |
| GRreqlutil | mgrltil | Sets the interlace mode of the next palette for subsequent read operations (Section 8.7.2 on page 315) |
| GRgetnluts | mggnluts | Retrieves the number of palettes associated with an image (See the *HDF Reference Manual*) |
| Miscenlaneous | GRsetcompress | mgscomp | Specifies whether the raster image will be stored in a file as a compressed raster image (Section 8.6.2 on page 312) |
| GRgetcompinfo | mggcompress | Retrieves image compression type and compression information (Section 8.9.6 on page 323) |
| GRsetexternalfile | mgsxfil | Specifies that the raster image will be written to an external file (Section 8.6.4.1 on page 313) |
| GRsetaccesstype | nmgsactp | Sets the access for an RI to be either serial or parallel I/O () |
| Inquiry | GRattrinfo | mgatinf | Retrieves information about an attribute (Section 8.10.3 on page 332) |
| GRfindattr | mgfndat | Finds the index of a data object's attribute given an attribute name (Section 8.10.3 on page 332) |
| GRfileinfo | mgfinfo | Retrieves the number of raster images and the number of global attributes in the file (Section 8.9.1 on page 321) |
| GRgetiminfo | mggiinf | Retrieves general information about a raster image (Section 8.9.2 on page 322) |
| GRgetlutinfo | mgglinf | Retrieves information about a palette (Section 8.11.4 on page 339) |
| Chunking | GRsetchunk | mgschnk | Creates chunked raster image (Section 8.12.2 on page 350) |
| GRgetchunkinfo | mggichnk | Retrieves information about a chunked raster image (Section 8.12.5 on page 360) |
| GRsetchunkcache | mgscchnk | Sets maximum number of chunks to be cached (Section 8.12.6 on page 360) |
| GRreadchunk | mgrchnk/  mgrcchnk | Reads a data chunk from a chunked raster image (pixel-interlace only) (Section 8.12.4 on page 358) |
| GRwritechunk | mgwchnk/  mgwcchnk | Writes a data chunk to a chunked raster image (pixel-interlace only) (Section 8.12.3 on page 351) |

## Header Files Required by the GR Interface

The header file "hdf.h" must be included in any program that utilizes GR interface routines.

## 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 raster image is as follows:

* Open an HDF file.
* Initialize the GR interface.
* Open an existing raster image or create a new raster image.
* Perform desired operations on the raster image.
* Terminate access to the raster image.
* Terminate access to the GR interface by disposing of the interface identifier.
* Close the HDF 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, n\_dds\_block);

gr\_id = GRstart(file\_id);

ri\_id = GRselect(gr\_id, ri\_index);

OR ri\_id = GRcreate(gr\_id, name, n\_comps, data\_type, interlace\_mode, dim\_sizes);

<Optional operations>

status = GRendaccess(ri\_id);

status = GRend(gr\_id);

status = Hclose(file\_id);

FORTRAN: file\_id = hopen(filename, access\_mode, n\_dds\_block)

gr\_id = mgstart(file\_id)

ri\_id = mgselct(gr\_id, ri\_index)

OR ri\_id = mgcreat(gr\_id, name, n\_comps, data\_type, interlace\_mode, dim\_sizes)

<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 interface identifier for each file to be opened. Similarly, to access more than one raster image, a calling program must obtain a separate data set identifier for each data set.

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

### Accessing Images and Files: GRstart, GRselect, and GRcreate

In the GR interface, Hopen opens the files and GRstart initiates the GR interface. Note the contrast to the SD interface, where SDstart performs both tasks. For information on the use of Hopen, refer to Chapter 2, HDF Fundamentals*.* For information on SDstart, refer to Chapter 3, 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, file\_id, the file identifier returned by Hopen, and returns the interface identifier gr\_id or FAIL (or -1) upon unsuccessful completion. Hopen and GRstart can be called several times to access more than one file.

GRselect specifies the given image as the current image to be accessed. It takes two arguments, the GR interface identifier gr\_id and the raster image index ri\_index, and returns the raster image identifier ri\_id or FAIL (or -1) upon unsuccessful completion. The GR interface identifier is returned by GRstart. The raster image index specifies the position of the image relative to the beginning of the file; it is zero-based, meaning that the index of the first image in the file is 0. The index of a raster image can be obtained from the image’s name using the routine GRnametoindex or from the image’s reference number using GRreftoindex. These routines are discussed in Section 8.9.4 on page 323 and Section 8.9.5 on page 323. The index value must be less than the total number of raster images in the file; that number can be obtained using GRfileinfo, described in Section 8.9.1 on page 321.

The parameters for GRstart and GRselect are further defined in (See Table 8B on page 302).

GRcreate defines a new raster image using the arguments gr\_id, name, n\_comps, data\_type, interlace\_mode, 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 write the image to the file; this occurs only when GRendaccess is called. Thus, failing to call GRendaccess properly will cause a loss of data.

The buffer name contains the name of the image; it must not exceed H4\_MAX\_GR\_NAME (or 256). The parameter n\_comps specifies the number of pixel components in the raster image; it must have a value of at least 1. The parameter data\_type specifies the data type of the image data; it can be any of the data types supported by the HDF library. The HDF supported data type are defined in the header file “hntdefs.h” and listed in (See Table 2F on page 14).

The parameter interlace\_mode specifies the interlacing in which the raster image is to be written; it can be set to either MFGR\_INTERLACE\_PIXEL (or 0), MFGR\_INTERLACE\_LINE (or 1), or MFGR\_INTERLACE\_COMPONENT (or 2). 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 on page 281 of Chapter 7, 24-bit Raster Images (DF24 API). Component interlacing, as the name implies, describes interlacing raster data by color component. Note that images created with the GR interface are actually written to disk in pixel interlace mode; any user-specified interlace mode is stored in the file with the image and the image is automatically converted to that mode when it is read with a GR interface function.

The parameter dim\_sizes specifies the size of the two dimensions of the image. The dimension sizes must be specified; their values must be at least 1.

GRcreate returns the value of the raster image identifier if successful or FAIL (or -1) otherwise. The parameters for GRstart, GRselect, and GRcreate are further defined in (See TABLE 8B).

### Terminating Access to Images and Files: GRendaccess and GRend

GRendaccess disposes of the raster image identifier ri\_id and terminates access to the data set initiated by the corresponding call to GRselect or GRcreate. The calling program must make one GRendaccess call for every GRselect or GRcreate call made during its execution. Failing to call GRendaccess for each call to GRselect or GRcreate may result in a loss of data.

GRend disposes of the GR interface identifier gr\_id and terminates 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; failing to call GRend for each GRstart may result in a loss of data.

GRendaccess and GRend return SUCCEED (or 0) or FAIL (or -1). The parameters of these routines are further defined in Table 8B.

Hclose terminates access to an HDF file and should only be called after GRend has been called properly. Refer to Chapter 2, HDF Fundamentals, for a description of Hclose.

* GRstart, GRselect, GRcreate, GRendaccess, and GRend, Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRstart  [int32]  (mgstart) | file\_id | int32 | integer | File identifier |
| GRselect  [int32]  (mgselct) | gr\_id | int32 | integer | GR interface identifier |
| ri\_index | int32 | integer | Position of the raster image within the file |
| GRcreate  [int32]  (mgcreat) | gr\_id | int32 | integer | GR interface identifier |
| name | char \* | character\*(\*) | Name of the image |
| n\_comps | int32 | integer | Number of components in each pixel |
| data\_type | int32 | integer | Data type of the pixel component |
| 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 |
| GRendaccess  [intn]  (mgendac) | ri\_id | int32 | integer | Raster image identifier |
| GRend  [intn]  (mgend) | gr\_id | int32 | integer | GR interface identifier |

## Writing Raster Images

A raster image can be written partially or entirely. Partial writing includes writing to a contiguous region of the image and writing to selected locations in the image according to patterns defined by the user. This section describes the routine GRwriteimage and how it can write data to part of an image or to an entire image. The section also illustrates the concepts of compressing raster images and the use of external files to store image data.

### Writing Raster Images: GRwriteimage

GRwriteimage is used to either completely or partially fill an image array.

Writing data to an image array involves the following steps:

* Open a file and initialize the GR interface.
* Select an existing raster image or create a new one.
* Write data to the image array.
* Terminate access to the raster image.
* Terminate access to the GR interface and close the file.

The calling program must contain the following sequence of calls:

C: file\_id = Hopen(filename, access\_mode, num\_dds\_block);

gr\_id = GRstart(file\_id);

ri\_id = GRselect(gr\_id, ri\_index);

OR ri\_id = GRcreate(gr\_id, name, n\_comps, number\_type, interlace\_mode, dim\_sizes);

status = GRwriteimage(ri\_id, start, stride, edges, data);

status = GRendaccess(gr\_id);

status = GRend(ri\_id);

status = Hclose(file\_id);

FORTRAN: file\_id = hopen(filename, access\_mode, num\_dds\_block)

gr\_id = mgstart(file\_id)

ri\_id = mgselct(gr\_id, ri\_index);

OR ri\_id = mgcreat(gr\_id, name, n\_comps, number\_type, interlace\_mode, dim\_sizes);

status = mgwrimg(ri\_id, start, stride, edges, data)

OR status = mgwrcmg(ri\_id, start, stride, edges, data)

status = mgendac(ri\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

As with SD arrays, whole raster images, subsamples, and slabs can be written. The data to be written is defined by the values of the parameters start, stride, and edges, which correspond to the coordinate location of the data origin, number of values to be skipped along each dimension during write operation, and number of elements to be written along each dimension.

The array start specifies the starting location of the data to be written. Valid values of each element in the array start are 0 to the size of the corresponding raster image dimension - 1. The first element of the array start specifies an offset from the beginning of the array data along the fastest-changing dimension, which is the second dimension in C and the first dimension in FORTRAN-77. The second element of the array start specifies an offset from the beginning of the array data along the second fastest-changing dimension, which is the first dimension in C and the second dimension in FORTRAN-77. For example, if the first value of the array start is 2 and the second value is 3, the starting location of the data to be written is at the fourth row and third column in C, and at the third row and fourth column in FORTRAN-77. Note that the correspondence between elements in the array start and the raster image dimensions in the GR interface is different from that in the SD interface. See Section 3.6 on page 58on SDreaddata for an example of this.

The array stride specifies the writing pattern along each dimension. For example, if one of the elements of the array stride is 1, then every element along the corresponding dimension of the array data will be written. If one of the elements of the stride array is 2, then every other element along the corresponding dimension of the array data will be written, and so on. The correspondence between elements of the array stride and the dimensions of the array data is the same as described above for the array start.

Note that the FORTRAN-77 version of GRwriteimage has two routines; mgwrimg writes buffered numeric data and mgwcimg writes buffered character data.

GRwriteimage returns either SUCCEED (or 0) or FAIL (or -1). The parameters for GRwriteimage are described in Table 8C.

* GRwriteimage Parameter List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRwriteimage  [intn]  (mgwrimg/  mgwcimg) | ri\_id | int32 | integer | Raster image identifier returned by **GRcreate** |
| start | int32 [2] | integer (2) | Array containing the x,y-coordinate location where 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 |
| edges | int32 [2] | integer (2) | Array containing the number of data elements that will be written along each dimension |
| data | VOIDP | <valid numeric data type>(\*)/  character(\*) | Buffer for the image data to be written |

* Creating and Writing a Raster Image

This example illustrates the use of the routines Hopen/hopen, GRstart/mgstart, GRcreate/mgcreat, GRwriteimage/mgwrimg, GRendaccess/mgendac, GRend/mgend, and Hclose/hclose to create an HDF file and store a raster image in it.

In this example, the program creates the HDF file called "General\_RImages.hdf" and a raster image in the file. The image created is of size 5x10 and named "Image Array 1", and has data of the int16 data type, 2 components, and interlace mode MFGR\_INTERLACE\_PIXEL. Then the program writes the image data, terminates access to the image and the GR interface, and closes the file.

C version

C:

#include "hdf.h"

#define FILE\_NAME "General\_RImages.hdf"

#define IMAGE\_NAME "Image Array 1"

#define X\_LENGTH 10 /\* number of columns in the image \*/

#define Y\_LENGTH 5 /\* number of rows in the image \*/

#define N\_COMPS 2 /\* number of components in the image \*/

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 file\_id, /\* HDF file identifier \*/

gr\_id, /\* GR interface identifier \*/

ri\_id, /\* raster image identifier \*/

start[2], /\* start position to write for each dimension \*/

edges[2], /\* number of elements to be written

along each dimension \*/

dim\_sizes[2], /\* dimension sizes of the image array \*/

interlace\_mode, /\* interlace mode of the image \*/

data\_type, /\* data type of the image data \*/

i, j;

int16 image\_buf[Y\_LENGTH][X\_LENGTH][N\_COMPS];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Create and open the file.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_CREATE, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Set the data type, interlace mode, and dimensions of the image.

\*/

data\_type = DFNT\_INT16;

interlace\_mode = MFGR\_INTERLACE\_PIXEL;

dim\_sizes[0] = X\_LENGTH;

dim\_sizes[1] = Y\_LENGTH;

/\*

\* Create the raster image array.

\*/

ri\_id = GRcreate (gr\_id, IMAGE\_NAME, N\_COMPS, data\_type,

interlace\_mode, dim\_sizes);

/\*

\* Fill the image data buffer with values.

\*/

for (i = 0; i < Y\_LENGTH; i++)

{

for (j = 0; j < X\_LENGTH; j++)

{

image\_buf[i][j][0] = (i + j) + 1; /\* first component \*/

image\_buf[i][j][1] = (i + j) + 1; /\* second component \*/

}

}

/\*

\* Define the size of the data to be written, i.e., start from the origin

\* and go as long as the length of each dimension.

\*/

start[0] = start[1] = 0;

edges[0] = X\_LENGTH;

edges[1] = Y\_LENGTH;

/\*

\* Write the data in the buffer into the image array.

\*/

status = GRwriteimage(ri\_id, start, NULL, edges, (VOIDP)image\_buf);

/\*

\* Terminate access to the raster image and to the GR interface and,

\* close the HDF file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program create\_raster\_image

implicit none

C

C Parameter declaration

C

character\*19 FILE\_NAME

character\*13 IMAGE\_NAME

integer X\_LENGTH

integer Y\_LENGTH

integer N\_COMPS

C

parameter (FILE\_NAME = ’General\_RImages.hdf’,

+ IMAGE\_NAME = ’Image Array 1’,

+ X\_LENGTH = 10,

+ Y\_LENGTH = 5,

+ N\_COMPS = 2)

integer DFACC\_CREATE, DFNT\_INT16, MFGR\_INTERLACE\_PIXEL

parameter (DFACC\_CREATE = 4,

+ DFNT\_INT16 = 22,

+ MFGR\_INTERLACE\_PIXEL = 0)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgcreat, mgwrimg, mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer status

integer file\_id

integer gr\_id, ri\_id, num\_type, interlace\_mode

integer start(2), stride(2), edges(2), dimsizes(2)

integer i, j, k

integer\*2 image\_buf(N\_COMPS, X\_LENGTH, Y\_LENGTH)

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Create and open the file.

C

file\_id = hopen(FILE\_NAME, DFACC\_CREATE, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Set the number type, interlace mode, and dimensions of the image.

C

num\_type = DFNT\_INT16

interlace\_mode = MFGR\_INTERLACE\_PIXEL

dimsizes(1) = X\_LENGTH

dimsizes(2) = Y\_lENGTH

C

C Create the raster image array.

C

ri\_id = mgcreat(gr\_id, IMAGE\_NAME, N\_COMPS, num\_type,

+ interlace\_mode, dimsizes)

C

C Fill the image data buffer with values.

C

do 30 i = 1, Y\_LENGTH

do 20 j = 1, X\_LENGTH

do 10 k = 1, N\_COMPS

image\_buf(k,j,i) = (i+j) - 1

10 continue

20 continue

30 continue

C

C Define the size of the data to be written, i.e., start from the origin

C and go as long as the length of each dimension.

C

start(1) = 0

start(2) = 0

edges(1) = X\_LENGTH

edges(2) = Y\_LENGTH

stride(1) = 1

stride(2) = 1

C

C Write the data in the buffer into the image array.

C

status = mgwrimg(ri\_id, start, stride, edges, image\_buf)

C

C Terminate access to the raster image and to the GR interface,

C and close the HDF file.

C

status = mgendac(ri\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

end

* Modifying an Existing Raster Image

This example illustrates the use of the routines GRselect/mgselct to obtain an existing raster image and GRwrite/mgwrimg to modify image data.

In this example, the program selects the only raster image in the file "General\_RImages.hdf" created and written in Example 1, and modifies image data. The program also creates another raster image that is named "Image Array 2" and has 3 components with dimension size of 4x6, data type of DFNT\_CHAR8, and interlace mode of MFGR\_INTERLACE\_PIXEL.

C version

C:

#include "hdf.h"

#define FILE\_NAME "General\_RImages.hdf"

#define X1\_LENGTH 5 /\* number of columns in the first image

being modified \*/

#define Y1\_LENGTH 2 /\* number of rows in the first image

being modified \*/

#define N1\_COMPS 2 /\* number of components in the first image \*/

#define IMAGE1\_NAME "Image Array 1"

#define IMAGE2\_NAME "Image Array 2"

#define X2\_LENGTH 6 /\* number of columns in the second image \*/

#define Y2\_LENGTH 4 /\* number of rows in the second image \*/

#define N2\_COMPS 3 /\* number of components in the second image \*/

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 file\_id, /\* HDF file identifier \*/

gr\_id, /\* GR interface identifier \*/

ri1\_id, /\* raster image identifier \*/

start1[2], /\* start position to write for each dimension \*/

edges1[2], /\* number of elements to be written along

each dimension \*/

ri2\_id, /\* raster image identifier \*/

start2[2], /\* start position to write for each dimension \*/

edges2[2], /\* number of elements to be written along

each dimension \*/

dims\_sizes[2], /\* sizes of the two dimensions of the image array \*/

data\_type, /\* data type of the image data \*/

interlace\_mode; /\* interlace mode of the image \*/

int16 i, j; /\* indices for the dimensions \*/

int16 image1\_buf[Y1\_LENGTH][X1\_LENGTH][N1\_COMPS]; /\* data of first image \*/

char image2\_buf[Y2\_LENGTH][X2\_LENGTH][N2\_COMPS]; /\* data of second image\*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the HDF file for writing.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_WRITE, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Select the first raster image in the file.

\*/

ri1\_id = GRselect (gr\_id, 0);

/\*

\* Fill the first image data buffer with values.

\*/

for (i = 0; i < Y1\_LENGTH; i++)

{

for (j = 0; j < X1\_LENGTH; j++)

{

image1\_buf[i][j][0] = 0; /\* first component \*/

image1\_buf[i][j][1] = 0; /\* second component \*/

}

}

/\*

\* Define the size of the data to be written, i.e., start from the origin

\* and go as long as the length of each dimension.

\*/

start1[0] = start1[1] = 0;

edges1[0] = X1\_LENGTH;

edges1[1] = Y1\_LENGTH;

/\*

\* Write the data in the buffer into the image array.

\*/

status = GRwriteimage (ri1\_id, start1, NULL, edges1, (VOIDP)image1\_buf);

/\*

\* Set the interlace mode and dimensions of the second image.

\*/

data\_type = DFNT\_CHAR8;

interlace\_mode = MFGR\_INTERLACE\_PIXEL;

dims\_sizes[0] = X2\_LENGTH;

dims\_sizes[1] = Y2\_LENGTH;

/\*

\* Create the second image in the file.

\*/

ri2\_id = GRcreate (gr\_id, IMAGE2\_NAME, N2\_COMPS, data\_type,

interlace\_mode, dims\_sizes);

/\*

\* Fill the second image data buffer with values.

\*/

for (i = 0; i < Y2\_LENGTH; i++)

{

for (j = 0; j < X2\_LENGTH; j++)

{

image2\_buf[i][j][0] = ’A’; /\* first component \*/

image2\_buf[i][j][1] = ’B’; /\* second component \*/

image2\_buf[i][j][2] = ’C’; /\* third component \*/

}

}

/\*

\* Define the size of the data to be written, i.e., start from the origin

\* and go as long as the length of each dimension.

\*/

for (i = 0; i < 2; i++) {

start2[i] = 0;

edges2[i] = dims\_sizes[i];

}

/\*

\* Write the data in the buffer into the second image array.

\*/

status = GRwriteimage (ri2\_id, start2, NULL, edges2, (VOIDP)image2\_buf);

/\*

\* Terminate access to the raster images and to the GR interface, and

\* close the HDF file.

\*/

status = GRendaccess (ri1\_id);

status = GRendaccess (ri2\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program modify\_image

implicit none

C

C Parameter declaration

C

character\*19 FILE\_NAME

character\*13 IMAGE1\_NAME

integer X1\_LENGTH

integer Y1\_LENGTH

integer N1\_COMPS

character\*13 IMAGE2\_NAME

integer X2\_LENGTH

integer Y2\_LENGTH

integer N2\_COMPS

C

parameter (FILE\_NAME = ’General\_RImages.hdf’,

+ IMAGE1\_NAME = ’Image Array 1’,

+ IMAGE2\_NAME = ’Image Array 2’,

+ X1\_LENGTH = 5,

+ Y1\_LENGTH = 2,

+ N1\_COMPS = 2,

+ X2\_LENGTH = 6,

+ Y2\_LENGTH = 4,

+ N2\_COMPS = 3)

integer DFACC\_WRITE, DFNT\_INT16, DFNT\_CHAR8,

+ MFGR\_INTERLACE\_PIXEL

parameter (DFACC\_WRITE = 2,

+ DFNT\_CHAR8 = 4,

+ DFNT\_INT16 = 22,

+ MFGR\_INTERLACE\_PIXEL = 0)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgselct, mgcreat, mgwrimg, mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer status

integer file\_id

integer gr\_id, ri1\_id, ri2\_id, data\_type, interlace\_mode

integer start1(2), stride1(2), edges1(2)

integer start2(2), stride2(2), edges2(2), dim\_sizes(2)

integer i, j, k

integer\*2 image1\_buf(N1\_COMPS, X1\_LENGTH, Y1\_LENGTH)

character image2\_buf(N2\_COMPS, X2\_LENGTH, Y2\_LENGTH)

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Open the HDF file for writing.

C

file\_id = hopen(FILE\_NAME, DFACC\_WRITE, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Select the first raster image in the file.

C

ri1\_id = mgselct(gr\_id, 0)

C

C Fill the buffer with values.

C

do 20 i = 1, Y1\_LENGTH

do 10 j = 1, X1\_LENGTH

image1\_buf(1,j,i) = 0

image1\_buf(2,j,i) = 0

10 continue

20 continue

C

C Define the part of the data in the first image that will be overwritten

C with the new values from image1\_buf.

C

start1(1) = 0

start1(2) = 0

edges1(1) = X1\_LENGTH

edges1(2) = Y1\_LENGTH

stride1(1) = 1

stride1(2) = 1

C

C Write the data in the buffer into the image array.

C

status = mgwrimg(ri1\_id, start1, stride1, edges1, image1\_buf)

C

C Set the number type, interlace mode, and dimensions of the second image.

C

data\_type = DFNT\_CHAR8

interlace\_mode = MFGR\_INTERLACE\_PIXEL

dim\_sizes(1) = X2\_LENGTH

dim\_sizes(2) = Y2\_LENGTH

C

C Create the second image in the file.

C

ri2\_id = mgcreat(gr\_id, IMAGE2\_NAME, N2\_COMPS, data\_type,

+ interlace\_mode, dim\_sizes)

C

C Fill the image data buffer with values.

C

do 60 i = 1, Y2\_LENGTH

do 50 j = 1, X2\_LENGTH

do 40 k = 1, N2\_COMPS

image2\_buf(k,j,i) = char(65 + k - 1)

40 continue

50 continue

60 continue

C

C Define the size of the data to be written, i.e., start from the origin

C and go as long as the length of each dimension.

C

start2(1) = 0

start2(2) = 0

edges2(1) = dim\_sizes(1)

edges2(2) = dim\_sizes(2)

stride2(1) = 1

stride2(2) = 1

C

C Write the data in the buffer into the image array.

C

status = mgwrimg(ri2\_id, start2, stride2, edges2, image2\_buf)

C

C Terminate access to the raster images and to the GR interface,

C and close the HDF file.

C

status = mgendac(ri1\_id)

status = mgendac(ri2\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

end

### Compressing Raster Images: GRsetcompress

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

C: status = GRsetcompress(ri\_id, comp\_type, c\_info);

FORTRAN: status = mgscompress(ri\_id, comp\_type, comp\_prm)

The compression method is specified by the parameter comp\_type. Valid values of the parameter comp\_type are:

COMP\_CODE\_NONE (or 0) for no compression

COMP\_CODE\_RLE (or 1) for RLE run-length encoding

COMP\_CODE\_SKPHUFF (or 3) for Skipping Huffman compression

COMP\_CODE\_DEFLATE (or 4) for GZIP compression

COMP\_CODE\_SZIP (or 5) for Szip compression (not for Fortran)

COMP\_CODE\_JPEG (or 7) for JPEG compression

The compression parameters are specified by the parameter c\_info in C and the parameter comp\_prm in FORTRAN-77. The parameter c\_info has type comp\_info and contains algorithm-specific information for the library compression routines. The type comp\_info is described in the header file hcomp.h and in the reference manual page for SDsetcompress. Compression parameters are only needed when Skipping Huffman, GZIP, and Szip compression methods are applied.

If comp\_type is set to COMP\_CODE\_NONE or COMP\_CODE\_RLE, the parameters c\_info and comp\_prm are not used; c\_info can be set to NULL and comp\_prm can be undefined.

If comp\_type is set to COMP\_CODE\_SKPHUFF, then the structure skphuff in the union comp\_info in C (comp\_prm(1) in FORTRAN-77) must be provided with the size, in bytes, of the data elements.

If comp\_type is set to COMP\_CODE\_DEFLATE, the deflate structure in the union comp\_info in C (comp\_prm(1) in FORTRAN-77) must be provided with the information about the compression effort.

Note that, as of HDF 4.2.2, Szip is not supported in Fortran GR interface yet.

GRsetcompress returns either SUCCEED (or 0) or FAIL (or -1). The GRsetcompress parameters are further described in Table 8D.

### Setting I/O Access Type for a Raster Image: GRsetaccesstype

GRsetaccesstype sets the access type to be either serial or parallel I/O for the raster image specified by *ri\_id*.

The syntax of the routine GRsetaccesstype is as follows:

C: status = GRsetaccesstype(ri\_id, access\_type);

FORTRAN: status = mgsactp(ri\_id, access\_type)

The access type is specified by the parameter access\_type and its valid values are DFACC\_SERIAL (or 1), DFACC\_PARALLEL (or 11), and DFACC\_DEFAULT (or 0.)

GRsetaccesstype returns either SUCCEED (or 0) or FAIL (or -1). The GRsetaccesstype parameters are further described in Table 8D.

* GRsetcompress and GRsetaccesstype Parameter List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRsetcompress  [intn]  (mgscompress**)** | ri\_id | int32 | integer | Raster image identifier |
| comp\_type | int32 | integer | Compression method |
| c\_info | comp\_info\* | N/A | Pointer to compression information structure |
| comp\_prm | N/A | integer | Compression parameters array |
| GRsetaccesstype  [intn]  (mgsactp**)** | ri\_id | int32 | integer | Raster image identifier |
| access\_type | int32 | integer | I/O access type |

### External File Operations Using the GR Interface

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

#### Creating a Raster Image in an External File: GRsetexternalfile

Creating an image with the data stored in an external file involves the same general steps as with the SD interface:

* Create the image array.
* Specify that an external data file is to be used.
* Write data to the image array.
* Terminate access to the image.

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

C: ri\_id = GRcreate(gr\_id, name, n\_comps, data\_type, interlace\_mode, dim\_sizes);

status = GRsetexternalfile(ri\_id, filename, offset);

status = GRwriteimage(ri\_id, start, stride, edges, image\_data);

status = GRendaccess(ri\_id);

FORTRAN: ri\_id = mgcreat(gr\_id, name, n\_comps, data\_type, interlace\_mode, dim\_sizes)

status = mgsxfil(ri\_id, filename, offset)

status = mgwrimg(ri\_id, start, stride, edges, image\_data)

status = mgendac(ri\_id)

GRsetexternalfile marks the image identified by the parameter ri\_id as one whose data is to be written to an external file. The parameter filename is the name of the external file, and the parameter offset specifies 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 name of the external file is specified, it is impossible to change it without breaking the association between the raster image and its data.

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

GRsetexternalfile returns either SUCCEED (or 0) or FAIL (or -1). The parameters of GRsetexternalfile are further defined in Table 8E.

* GRsetexternalfile Parameter List

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRsetexternalfile  [intn]  (mgsxfil) | ri\_id | int32 | integer | Raster image identifier |
| filename | char \* | character\*(\*) | Name of the external file |
| offset | int32 | integer | Offset in bytes from the beginning of the external file to the image data |

#### Moving Raster Images to an External File

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

* Select the image.
* Specify the external data file.
* Terminate access to the image.

The calling program must make the following calls:

C: ri\_id = GRselect(gr\_id, ri\_index);

status = GRsetexternalfile(ri\_id, filename, offset);

status = GRendaccess(ri\_id);

FORTRAN: ri\_id = mgselct(gr\_id, ri\_index);

status = mgsxfil(ri\_id, filename, offset)

status = mgendac(ri\_id);

When GRsetexternalfile is used in conjunction with GRselect, it will immediately write the existing data to the external file; 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.

## Reading Raster Images

Image array data can be read as an entire array or as a subsample of the array. Raster image data is read from an external file in the same way that it is read from a primary file; whether the image array is stored in an external file is transparent to the user. This section describes how GRreadimage is used to read an entire image and part of an image. The section also describes the routine GRreqimageil that sets the interlacing for reading image data.

### Reading Data from an Image: GRreadimage

Reading data subsamples from an image array involves the following steps:

* Select a data set.
* Read data from the image array.
* Terminate access to the data set.

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

C: ri\_id = GRselect(gr\_id, ri\_index);

status = GRreadimage(ri\_id, start, stride, edges, data);

status = GRendaccess(ri\_id);

FORTRAN: ri\_id = mgselct(gr\_id, ri\_index)

status = mgrdimg(ri\_id, start, stride, edges, data)

OR status = mgrcimg(ri\_id, start, stride, edges, data)

status = mgendac(gr\_id)

GRreadimage can be used to read either an entire image or a subsample of the image. The ri\_id argument is the raster image identifier returned by GRselect. As with GRwriteimage, the arguments start, stride, and edges 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. Refer to Section 8.6.1 on page 302 for detailed descriptions of the parameters start, stride, and edges. 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.

Note that the FORTRAN-77 version of GRreadimage has two routines; mgrdimg reads numeric image data and mgrcimg reads character image data.

GRreadimage returns either SUCCEED (or 0) or FAIL (or -1). The parameters for GRreadimage are further defined in (See TABLE 8F).

### Setting the Interlace Mode for an Image Read: GRreqimageil

The GRreqimageil routine sets the interlace mode for the next image read. The syntax of this routine is as follows:

C: status = GRreqimageil(ri\_id, interlace\_mode);

FORTRAN: status = mgrimil(ri\_id, interlace\_mode)

GRreqimageil can be called at any time before the read operation and takes two parameters, ri\_id and interlace\_mode. The parameter ri\_id is the raster image identifier returned by the GRselect routine and the parameter interlace\_mode specifies the interlace mode that will be in effect for the image read operation. Refer to Section 8.5.1 on page 300 for a description of the GR interlace modes.

GRreqimagetil may be called more than once; the interlace mode setting specified by the last call to the routine will be used for the next read operation.

GRreqimagetil returns either SUCCEED (or 0) or FAIL (or -1). The parameters of this routine are further defined in Table 8F.

* GRreadimage and GRreqimageil Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRreadimage  [intn]  (mgrdimg/  mgrcimg) | ri\_id | int32 | integer | Raster image identifier |
| start | int32[2] | integer (2) | Array containing the starting read coordinates |
| stride | int32[2] | integer (2) | Array specifying the interval between the values that will be read along each dimension |
| edges | int32[2] | integer (2) | Array containing the number of data elements that will be read along each dimension |
| data | VOIDP | <valid numeric data type>(\*)/character\*(\*) | Buffer for the image data to be read |
| GRreqimageil  [intn]  (mgrimil) | ri\_id | int32 | integer | Raster image identifier |
| interlace\_mode | intn | integer | Interlace mode for the next image read operation |

* Reading a Raster Image.

This example illustrates the use of the routine GRreadimage/mgrdimg to read an image and its subsets.

In this example, the program reads the image written by Example 1 and modified by Example 2 in the file "General\_RImages.hdf". Recall that this image has two components and has 5 rows and 10 columns. The program first reads the entire image, then reads a subset of the image, 3 rows and 2 columns starting at the 2nd row and the 4th column, and finally reads the image skipping all the even rows and all the odd columns. Reading patterns are applied to all components.

C version

C:

#include "hdf.h"

#define FILE\_NAME "General\_RImages.hdf"

#define N\_COMPS 2

#define X\_LENGTH 10 /\* number of columns of the entire image \*/

#define Y\_LENGTH 5 /\* number of rows of the entire image \*/

#define PART\_COLS 2 /\* number of columns read for partial image \*/

#define PART\_ROWS 3 /\* number of rows read for partial image \*/

#define SKIP\_COLS 5 /\* number of columns read for skipped image \*/

#define SKIP\_ROWS 3 /\* number of rows read for skipped image \*/

#define COLS\_PART\_START 3 /\* starting column to read partial image \*/

#define ROWS\_PART\_START 1 /\* starting row to read partial image \*/

#define COLS\_SKIP\_START 1 /\* starting column to read skipped image \*/

#define ROWS\_SKIP\_START 0 /\* starting row to read skipped image \*/

#define N\_STRIDES 2 /\* number of elements to skip on each dim. \*/

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 index;

int32 file\_id, gr\_id, ri\_id,

start[2], /\* start position to write for each dimension \*/

edges[2], /\* number of elements to bewritten along

each dimension \*/

stride[2], /\* number of elements to skip on each dimension \*/

dim\_sizes[2]; /\* dimension sizes of the image array \*/

int16 entire\_image[Y\_LENGTH][X\_LENGTH][N\_COMPS],

partial\_image[PART\_ROWS][PART\_COLS][N\_COMPS],

skipped\_image[SKIP\_ROWS][SKIP\_COLS][N\_COMPS];

int32 i, j;

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the HDF file for reading.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_READ, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Select the first raster image in the file.

\*/

ri\_id = GRselect (gr\_id, 0);

/\*

\* Define the size of the data to be read, i.e., start from the origin

\* and go as long as the length of each dimension.

\*/

start[0] = start[1] = 0;

edges[0] = X\_LENGTH;

edges[1] = Y\_LENGTH;

/\*

\* Read the data from the raster image array.

\*/

status = GRreadimage (ri\_id, start, NULL, edges, (VOIDP)entire\_image);

/\*

\* Display only the first component of the image since the two components

\* have the same data in this example.

\*/

printf ("First component of the entire image:\n");

for (i = 0; i < Y\_LENGTH; i++)

{

for (j = 0; j < X\_LENGTH; j++)

printf ("%d ", entire\_image[i][j][0]);

printf ("\n");

}

/\*

\* Define the size of the data to be read.

\*/

start[0] = COLS\_PART\_START;

start[1] = ROWS\_PART\_START;

edges[0] = PART\_COLS;

edges[1] = PART\_ROWS;

/\*

\* Read a subset of the raster image array.

\*/

status = GRreadimage (ri\_id, start, NULL, edges, (VOIDP)partial\_image);

/\*

\* Display the first component of the read sample.

\*/

printf ("\nThree rows & two cols at 2nd row and 4th column");

printf (" of the first component:\n");

for (i = 0; i < PART\_ROWS; i++)

{

for (j = 0; j < PART\_COLS; j++)

printf ("%d ", partial\_image[i][j][0]);

printf ("\n");

}

/\*

\* Define the size and the pattern to read the data.

\*/

start[0] = COLS\_SKIP\_START;

start[1] = ROWS\_SKIP\_START;

edges[0] = SKIP\_COLS;

edges[1] = SKIP\_ROWS;

stride[0] = stride[1] = N\_STRIDES;

/\*

\* Read all the odd rows and even columns of the image.

\*/

status = GRreadimage (ri\_id, start, stride, edges, (VOIDP)skipped\_image);

/\*

\* Display the first component of the read sample.

\*/

printf ("\nAll odd rows and even columns of the first component:\n");

for (i = 0; i < SKIP\_ROWS; i++)

{

for (j = 0; j < SKIP\_COLS; j++)

printf ("%d ", skipped\_image[i][j][0]);

printf ("\n");

}

/\*

\* Terminate access to the raster image and to the GR interface, and

\* close the HDF file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program read\_raster\_image

implicit none

C

C Parameter declaration

C

character\*19 FILE\_NAME

integer X\_LENGTH

integer Y\_LENGTH

integer N\_COMPS

C

parameter (FILE\_NAME = ’General\_RImages.hdf’,

+ X\_LENGTH = 10,

+ Y\_LENGTH = 5,

+ N\_COMPS = 2)

integer PART\_COLS, PART\_ROWS, SKIP\_COLS, SKIP\_ROWS

integer COLS\_PART\_START, ROWS\_PART\_START

integer COLS\_SKIP\_START, ROWS\_SKIP\_START

integer N\_STRIDES

parameter (PART\_COLS = 3, PART\_ROWS = 2,

+ SKIP\_COLS = 3, SKIP\_ROWS = 5,

+ COLS\_PART\_START = 1, ROWS\_PART\_START = 3,

+ COLS\_SKIP\_START = 0, ROWS\_SKIP\_START = 1,

+ N\_STRIDES = 2)

integer DFACC\_READ

parameter (DFACC\_READ = 1)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgselct, mgrdimg, mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer status

integer file\_id

integer gr\_id, ri\_id

integer start(2), stride(2), edges(2)

integer i, j

integer\*2 entire\_image(N\_COMPS, X\_LENGTH, Y\_LENGTH)

integer\*2 partial\_image(N\_COMPS, PART\_ROWS, PART\_COLS)

integer\*2 skipped\_image(N\_COMPS, SKIP\_ROWS, SKIP\_COLS)

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Open the HDF file for reading.

C

file\_id = hopen(FILE\_NAME, DFACC\_READ, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Select the first raster image in the file.

C

ri\_id = mgselct(gr\_id, 0)

C

C Define the size of the data to be read, i.e., start from the origin

C and go as long as the length of each dimension.

C

start(1) = 0

start(2) = 0

edges(1) = X\_LENGTH

edges(2) = Y\_LENGTH

stride(1) = 1

stride(2) = 1

C

C Read the data from the raster image array.

C

status = mgrdimg(ri\_id, start, stride, edges, entire\_image)

C

C Display only the first component of the image since the two components

C have the same data in this example.

C

write(\*,\*) ’First component of the entire image’

write(\*,\*)

do 10 i = 1, X\_LENGTH

write(\*,1000) (entire\_image(1,i,j), j = 1, Y\_LENGTH)

10 continue

write(\*,\*)

C

C Define the size of the data to be read.

C

start(1) = ROWS\_PART\_START

start(2) = COLS\_PART\_START

edges(1) = PART\_ROWS

edges(2) = PART\_COLS

stride(1) = 1

stride(2) = 1

C

C Read a subset of the raster image array.

C

status = mgrdimg(ri\_id, start, stride, edges, partial\_image)

C

C Display only the first component of the read sample.

C

write(\*,\*)

+ ’Two rows and three columns at 4th row and 2nd column’,

+ ’ of the first component’

write(\*,\*)

do 20 i = 1, PART\_ROWS

write(\*,1000) (partial\_image(1,i,j), j = 1, PART\_COLS)

20 continue

write(\*,\*)

C

C Define the size and the pattern to read the data.

C

start(1) = ROWS\_SKIP\_START

start(2) = COLS\_SKIP\_START

edges(1) = SKIP\_ROWS

edges(2) = SKIP\_COLS

stride(1) = N\_STRIDES

stride(2) = N\_STRIDES

C

C Read all the odd rows and even columns of the image.

C

status = mgrdimg(ri\_id, start, stride, edges, skipped\_image)

C

C Display only the first component of the read sample.

C

write(\*,\*) ’All even rows and odd columns of the first component’

write(\*,\*)

do 30 i = 1, SKIP\_ROWS

write(\*,1000) (skipped\_image(1,i,j), j = 1, SKIP\_COLS)

30 continue

write(\*,\*)

C

C Terminate access to the raster image and to the GR interface,

C and close the HDF file.

C

status = mgendac(ri\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

1000 format(1x, 5(I4))

end

## Difference between the SD and GR Interfaces

There is a difference between the SD and GR interfaces that becomes important in applications or tools that must manipulate both images and two-dimensional SDs.

The SD and GR interfaces differ in the correspondence between the dimension order in parameter arrays such as start, stride, edge, and dimsizes and the dimension order in the *data* array. See the SDreaddata and GRreadimage reference manual pages for discussions of the SD and GR approaches, respectively.

When writing applications or tools to manipulate both images and two-dimensional SDs, this crucial difference between the interfaces must be taken into account. While the underlying data is stored in row-major order in both cases, the API parameters are not expressed in the same way. Consider the example of an SD data set and a GR image that are stored as identically-shaped arrays of X columns by Y rows and accessed via the SDreaddata and GRreadimage functions, respectively. Both functions take the parameters start, stride, and edge.

* For SDreaddata, those parameters are expressed in (y,x) or [row,column] order. For example, start[0] is the starting point in the Y dimension and start[1] is the starting point in the X dimension. The same ordering holds true for all SD data set manipulation functions.
* For GRreadimage, those parameters are expressed in (x,y) or [column,row] order. For example, start[0] is the starting point in the X dimension and start[1] is the starting point in the Y dimension. The same ordering holds true for all GR functions manipulating image data.

## Obtaining Information about Files and Raster Images

The routines covered in this section provide methods for obtaining information about all of the images in a file, for identifying images that meet certain criteria, and for obtaining information about specific raster images.

GRfileinfo retrieves the number of images and file attributes in a file. GRgetiminfo provides information about individual images. To retrieve information about all images in a file, a calling program can use GRfileinfo to determine the number of images, followed by repeated calls to GRgetiminfo to obtain information about each image.

GRnametoindex or GRreftoindex can be used to obtain the index of a raster image in a file knowing its name or reference number, respectively. Refer to Section 8.2.1 on page 296 for a description of the raster image index and reference number. GRidtoref is used when the reference number of an image is required by another routine and the raster image identifier is available.

These routines are described individually in the following subsections.

### Obtaining Information about the Contents of a File: GRfileinfo

GRfileinfo retrieves the number of raster images and the number of file attributes contained in a file. This information is often useful in index validation, sequential searches, or memory allocation. The syntax of GRfileinfo is as follows:

C: status = GRfileinfo(gr\_id, &n\_images, &n\_file\_attrs);

FORTRAN: status = mgfinfo(gr\_id, n\_images, n\_file\_attrs)

The number of images in the file and the total number of file attributes will be stored in the arguments n\_images and n\_file\_attrs, respectively.

GRfileinfo returns SUCCEED (or 0) if successful or FAIL (or -1) otherwise. The parameters for GRfileinfo are further described in (See Table 8G on page 322).

### 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 routine GRgetiminfo retrieves this required information. To access information about an image, the calling program must contain the following:

C: status = GRgetiminfo(ri\_id, name, &n\_comps, &data\_type, &interlace\_mode, dim\_sizes, &n\_attrs);

FORTRAN: status = mggiinf(ri\_id, name, n\_comps, data\_type, interlace\_mode, dim\_sizes, n\_attrs)

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 arguments name, n\_comps, data\_type, interlace\_mode, dim\_sizes, and n\_attrs 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.

GRgetiminfo returns either SUCCEED (or 0) or FAIL (or -1). The parameters of this routine are further defined in Table 8G.

* GRfileinfo and GRgetiminfo Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRfileinfo  [intn]  (mgfinfo) | gr\_id | int32 | integer | GR interface identifier |
| n\_images | int32 \* | integer | Number of raster images in the file |
| n\_file\_attrs | int32 \* | integer | Number of global attributes in the file |
| GRgetiminfo  [intn]  (mggiinf) | ri\_id | int32 | integer | Raster image identifier |
| name | char \* | character\*(\*) | Name of the raster image |
| n\_comps | int32 \* | integer | Number of pixel components in the pixel |
| data\_type | int32 \* | integer | Pixel data type |
| interlace\_mode | int32 \* | integer | Interlace mode of the data in the raster image |
| dim\_sizes | int32 [2] | integer (2) | Array containing the size of each dimension in the raster image |
| n\_attrs | int32 \* | integer | Number of raster image attributes |

### Obtaining the Reference Number of a Raster Image from Its Identifier: GRidtoref

GRidtoref returns either the reference number of the raster image identified by the parameter ri\_id, or FAIL (or -1) upon unsuccessful completion. The syntax of GRidtoref is as follows:

C: ref = GRidtoref(ri\_id);

FORTRAN: ref = mgid2rf(ri\_id)

This routine is further defined in (See Table 8H on page 325).

### Obtaining the Index of a Raster Image from Its Reference Number: GRreftoindex

GRreftoindex returns either the index of the raster image specified by its reference number, ref, or FAIL (or -1) upon unsuccessful completion. The syntax of GRreftoindex is as follows:

C: ri\_index = GRreftoindex(gr\_id, ref);

FORTRAN: ri\_index = mgr2idx(gr\_id, ref)

This routine is further defined in Table 8H.

### Obtaining the Index of a Raster Image from Its Name: GRnametoindex

GRnametoindex returns the index of the raster image specified by its name or FAIL (or -1) upon unsuccessful completion. The syntax of GRnametoindex is as follows:

C: ri\_index = GRnametoindex(gr\_id, name);

FORTRAN: ri\_index = mgr2idx(gr\_id, name)

This routine is further defined in Table 8H.

### Obtaining Compression Information for a Raster Image: GRgetcompinfo

GRgetcompinfo retrieves the type of compression used to store a raster image and, when appropriate, the required compression parameters. GRgetcompinfo replaces GRgetcompress because this function has flaws, causing failure for some chunked and chunked/compressed data.

GRgetcompinfo takes one input parameter, ri\_id, a raster image identifier, and two output parameters, comp\_type, for the type of compression used when the image was written, and either c\_info (a C struct) or comp\_prm (a FORTRAN-77 array) for the returned compression parameters.

Valid comp\_type values are as follows:

COMP\_CODE\_NONE (or 0) for no compression

COMP\_CODE\_RLE (or 1) for RLE run-length encoding

COMP\_CODE\_SKPHUFF (or 3) for Skipping Huffman compression

COMP\_CODE\_DEFLATE (or 4) for GZIP compression

COMP\_CODE\_SZIP (or 5) for Szip compression (not for Fortran)

COMP\_CODE\_JPEG (or 7) for JPEG compression

The c\_info struct is of type comp\_info, contains algorithm-specific information for the library compression routines, and is described in the hcomp.h header file.

The comp\_prm parameter is an array of several elements.

For Skipping Huffman compression, comp\_prm(1) contains the skip value, skphuff\_skp\_size.

For GZIP compression, comp\_prm(1) contains the deflation value, deflate\_value.

For other compression types, comp\_prm is ignored. Currently, Szip is not yet supported in Fortran GR interface.

GRgetcompinfo returns SUCCESS (or 0) if it is successful or FAIL (or -1) upon unsuccessful completion.

The syntax of GRgetcompinfo is as follows:

C: status = GRgetcompinfo(ri\_id, comp\_type, c\_info);

FORTRAN: status = mggcompress(ri\_id, comp\_type, comp\_prm)

This routine is further defined in Table 8H.

### Checking Whether a Raster Image Is To Be Mapped: GR2bmapped

This function was originally added to support the HDF4 File Content Project. The tool, produced from the project, maps the contents of HDF4 files. Supporting for raster images was limited as requested by the project’s sponsor. Thus, only certain types of images, which satisfy a set of conditions, are to be mapped.

GR2bmapped will set tobe\_mapped to TRUE if the given raster image, ri\_id, satisfies the following conditions:

* being an 8-bit raster image,
* having one component,
* being non-special or RLE compressed only, i.e., no other compressions or chunking,

or FAIL (or -1), otherwise. The syntax of GR2bmapped is as follows:

C: status = GR2bmapped(ri\_id, &tobe\_mapped, &name\_generated);

FORTRAN: Unavailable

Another characteristic of the image to be reported by GR2bmapped is whether the image has name that was generated by the library and, if so, name\_generated will be set to TRUE. Old images (or images created with pre-GR API) do not have a name and the library would generate a name for it while reading in the file. The tool from the HDF4 File Content Project needs to make this distinction.

GR2bmapped returns SUCCEED (or 0), if successful, or FAIL (or -1), otherwise. When failure occurs, tobe\_mapped and name\_generated will be undefined. This routine is further defined in Table 8H.

* GRidtoref, GRreftoindex, GRnametoindex, and GRgetcompinfo Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRidtoref  [uint16]  (mgid2rf) | ri\_id | int32 | integer | Raster image identifier |
| GRreftoindex  [int32]  (mgr2idx) | gr\_id | int32 | integer | GR interface identifier |
| ref | uint16 | integer | Reference number of the raster image |
| GRnametoindex  [int32]  (mgn2ndx) | gr\_id | int32 | integer | GR interface identifier |
| name | char \* | character \*(\*) | Name of the raster image |
| GRgetcompinfo  [intn]  (mggcompress) | ri\_id | int32 | integer | Raster image identifier |
| comp\_type | comp\_coder\_t | integer | Type of compression |
| c\_info | comp\_info | N/A | Pointer to compression information structure |
| comp\_prm(1) | N/A | integer | Compression parameter in array format |
| GR2bmapped  [intn]  (unavailable) | ri\_id | int32 | integer | Raster image identifier |
| tobe\_mapped | intn \* | integer | TRUE if the image should be mapped |
| name\_generated | intn \* | N/A | TRUE if the image's name was generated by the GR API, i.e., not given by applications |

* Obtaining File and Image Information.

This example illustrates the use of the routines GRfileinfo/mgfinfo and GRgetiminfo/mggiinf to obtain information such as the number of images and attributes in an HDF file and the characteristics of a raster image in the file.

In this example, the program gets the number of images in the file using the routine GRfileinfo/mgfinfo. For each image, the program then obtains and displays its name, number of components, data type, interlace mode, dimension sizes, and number of attributes using the routine GRgetiminfo/mggiinf.

C version

C:

#include "hdf.h"

#define FILE\_NAME "General\_RImages.hdf"

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 file\_id, gr\_id, ri\_id,

n\_rimages, /\* number of raster images in the file \*/

n\_file\_attrs, /\* number of file attributes \*/

ri\_index, /\* index of a image \*/

dim\_sizes[2], /\* dimensions of an image \*/

n\_comps, /\* number of components an image contains \*/

interlace\_mode, /\* interlace mode of an image \*/

data\_type, /\* number type of an image \*/

n\_attrs; /\* number of attributes belong to an image \*/

char name[MAX\_GR\_NAME], /\* name of an image \*/

\*type\_string, /\* mapped text of a number type \*/

\*interlace\_string; /\* mapped text of an interlace mode \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the file for reading.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_READ, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Determine the contents of the file.

\*/

status = GRfileinfo (gr\_id, &n\_rimages, &n\_file\_attrs);

/\*

\* For each image in the file, get and display the image information.

\*/

printf ("RI# Name Components Type Interlace \

Dimensions Attributes\n\n");

for (ri\_index = 0; ri\_index < n\_rimages; ri\_index++)

{

ri\_id = GRselect (gr\_id, ri\_index);

status = GRgetiminfo (ri\_id, name, &n\_comps, &data\_type,

&interlace\_mode, dim\_sizes, &n\_attrs);

/\*

\* Map the number type and interlace mode into text strings for output

\* readability. Note that, in this example, only two possible types

\* are considered because of the simplicity of the example. For real

\* problems, all possible types should be checked and, if reading the

\* data is desired, the size of the type must be determined based on the

\* machine where the program resides.

\*/

if (data\_type == DFNT\_CHAR8)

type\_string = "Char8";

else if (data\_type == DFNT\_INT16)

type\_string = "Int16";

else

type\_string = "Unknown";

switch (interlace\_mode)

{

case MFGR\_INTERLACE\_PIXEL:

interlace\_string = "MFGR\_INTERLACE\_PIXEL";

break;

case MFGR\_INTERLACE\_LINE:

interlace\_string = "MFGR\_INTERLACE\_LINE";

break;

case MFGR\_INTERLACE\_COMPONENT:

interlace\_string = "MFGR\_INTERLACE\_COMPONENT";

break;

default:

interlace\_string = "Unknown";

break;

} /\* switch \*/

/\*

\* Display the image information for the current raster image.

\*/

printf ("%d %s %d %s %s %2d,%2d %d\n",

ri\_index, name, n\_comps, type\_string, interlace\_string,

dim\_sizes[0], dim\_sizes[1], n\_attrs);

/\*

\* Terminate access to the current raster image.

\*/

status = GRendaccess (ri\_id);

}

/\*

\* Terminate access to the GR interface and close the HDF file.

\*/

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program image\_info

implicit none

C

C Parameter declaration

C

character\*19 FILE\_NAME

C

parameter (FILE\_NAME = ’General\_RImages.hdf’)

integer DFACC\_READ

parameter (DFACC\_READ = 1)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgselct, mgfinfo, mggiinf, mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer status

integer file\_id, gr\_id, ri\_id

integer n\_rimages, n\_file\_attrs, ri\_index

integer n\_comps, interlace\_mode, n\_attrs, data\_type

integer dim\_sizes(2)

character\*10 type\_string

character\*24 interlace\_string

character\*64 name

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Open the HDF file for reading.

C

file\_id = hopen(FILE\_NAME, DFACC\_READ, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Determine the contents of the file.

C

status = mgfinfo(gr\_id, n\_rimages, n\_file\_attrs)

C

C For each image in the file, get and display image information.

C

do 100 ri\_index = 0, n\_rimages-1

ri\_id = mgselct(gr\_id, ri\_index)

status = mggiinf(ri\_id, name, n\_comps, data\_type,

+ interlace\_mode, dim\_sizes, n\_attrs)

C

C Map the number type and interlace mode into text strings for

C output readability.

C

if(data\_type .eq. 4) then

type\_string = ’DFNT\_CHAR8’

else if(data\_type .eq. 22) then

type\_string = ’DFNT\_INT16’

else

type\_string = ’Unknown’

endif

if (interlace\_mode .eq. 0) then

interlace\_string = ’MFGR\_INTERLACE\_PIXEL’

else if(interlace\_mode .eq. 1) then

interlace\_string = ’MFGR\_INTERLACE\_LINE’

else if(interlace\_mode .eq. 2) then

interlace\_string = ’MFGR\_INTERLACE\_COMPONENT’

else

interlace\_string = ’Unknown’

endif

C

C Display the image information for the current image.

C

write(\*,\*) ’Image index: ’, ri\_index

write(\*,\*) ’Image name: ’, name

write(\*,\*) ’Number of components: ’, n\_comps

write(\*,\*) ’Number type: ’, type\_string

write(\*,\*) ’Interlace mode: ’, interlace\_string

write(\*,\*) ’Dimnesions: ’, dim\_sizes(1), dim\_sizes(2)

write(\*,\*) ’Number of image attributes: ’, n\_attrs

write(\*,\*)

C

C Terminate access to the current raster image.

C

status = mgendac(ri\_id)

100 continue

C

C Terminate access to the GR interface and close the HDF file.

status = mgend(gr\_id)

status = hclose(file\_id)

end

## GR Data Set Attributes

The GR interface provides tools that attach attributes to particular images. This capability is similar to, though more limited than, attribute function capabilities of the SD interface. The concepts of user-defined and predefined attributes are explained in Chapter 3, 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.

### Predefined GR Attributes

The GR API 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, i.e., 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, GRsetattr, is explained in the next section.

### Setting User-defined Attributes: GRsetattr

GRsetattr creates or modifies an attribute for either a file or a raster image. If the attribute with the specified name does not exist, GRsetattr creates a new one. If the named attribute already exists, GRsetattr resets all the values that are different from those provided in its argument list. The syntax of GRsetattr is as follows:

C: status = GRsetattr(obj\_id, attr\_name, data\_type, n\_values, attr\_value);

FORTRAN: status = mgsnatt(obj\_id, attr\_name, data\_type, n\_values, attr\_value)

OR status = mgscatt(obj\_id, attr\_name, data\_type, n\_values, attr\_value)

The first argument, obj\_id, can either be the GR interface identifier or raster image identifier. The argument attr\_name contains the name of the attribute and can be no more than H4\_MAX\_GR\_NAME (or 256) characters in length. Passing the name of an existing attribute will overwrite the value portion of that attribute.

The arguments data\_type, n\_values, 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 n\_values contains the total number of values in the attribute.

Note that the FORTRAN-77 version of GRsetattr has two routines; mgsnatt writes numeric attribute data and mgscatt writes character attribute data.

GRsetattr returns either SUCCEED (or 0) or FAIL (or -1). The parameters for GRsetattr are further described in (See Table 8I on page 334).

* Operations on File and Raster Image Attributes.

This example illustrates the use of the routines GRsetattr/mgsnatt/mgscatt to assign attributes to an HDF file and to an image.

In this example, the program sets two attributes to the existing file "General\_RImages.hdf" and two attributes to the image named "Image Array 2". The file is created by the program in Example 1 and the image is created by the program in Example 2. The values of the second attribute of the image are of type int16 and the values of the other three attributes are of type char8.

C version

C:

#include "hdf.h"

#define FILE\_NAME "General\_RImages.hdf"

#define IMAGE\_NAME "Image Array 2"

#define F\_ATT1\_NAME "File Attribute 1"

#define F\_ATT2\_NAME "File Attribute 2"

#define RI\_ATT1\_NAME "Image Attribute 1"

#define RI\_ATT2\_NAME "Image Attribute 2"

#define F\_ATT1\_VAL "Contents of First FILE Attribute"

#define F\_ATT2\_VAL "Contents of Second FILE Attribute"

#define F\_ATT1\_N\_VALUES 32

#define F\_ATT2\_N\_VALUES 33

#define RI\_ATT1\_VAL "Contents of IMAGE’s First Attribute"

#define RI\_ATT1\_N\_VALUES 35

#define RI\_ATT2\_N\_VALUES 6

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 gr\_id, ri\_id, file\_id,

ri\_index;

int16 ri\_attr\_2[RI\_ATT2\_N\_VALUES] = {1, 2, 3, 4, 5, 6};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the HDF file.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_WRITE, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Set two file attributes to the file with names, data types, numbers of

\* values, and values of the attributes specified.

\*/

status = GRsetattr (gr\_id, F\_ATT1\_NAME, DFNT\_CHAR8, F\_ATT1\_N\_VALUES,

(VOIDP)F\_ATT1\_VAL);

status = GRsetattr (gr\_id, F\_ATT2\_NAME, DFNT\_CHAR8, F\_ATT2\_N\_VALUES,

(VOIDP)F\_ATT2\_VAL);

/\*

\* Obtain the index of the image named IMAGE\_NAME.

\*/

ri\_index = GRnametoindex (gr\_id, IMAGE\_NAME);

/\*

\* Obtain the identifier of this image.

\*/

ri\_id = GRselect (gr\_id, ri\_index);

/\*

\* Set two attributes to the image with names, data types, numbers of

\* values, and values of the attributes specified.

\*/

status = GRsetattr (ri\_id, RI\_ATT1\_NAME, DFNT\_CHAR8, RI\_ATT1\_N\_VALUES,

(VOIDP)RI\_ATT1\_VAL);

status = GRsetattr (ri\_id, RI\_ATT2\_NAME, DFNT\_INT16, RI\_ATT2\_N\_VALUES,

(VOIDP)ri\_attr\_2);

/\*

\* Terminate access to the image and to the GR interface, and close the

\* HDF file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program set\_attribute

implicit none

C

C Parameter declaration

C

character\*19 FILE\_NAME

character\*13 IMAGE\_NAME

character\*16 F\_ATT1\_NAME

character\*16 F\_ATT2\_NAME

character\*17 RI\_ATT1\_NAME

character\*17 RI\_ATT2\_NAME

character\*32 F\_ATT1\_VAL

character\*33 F\_ATT2\_VAL

integer F\_ATT1\_N\_VALUES

integer F\_ATT2\_N\_VALUES

character\*35 RI\_ATT1\_VAL

integer RI\_ATT1\_N\_VALUES

integer RI\_ATT2\_N\_VALUES

C

parameter (FILE\_NAME = ’General\_RImages.hdf’,

+ IMAGE\_NAME = ’Image Array 2’,

+ F\_ATT1\_NAME = ’File Attribute 1’,

+ F\_ATT2\_NAME = ’File Attribute 2’,

+ RI\_ATT1\_NAME = ’Image Attribute 1’,

+ RI\_ATT2\_NAME = ’Image Attribute 2’,

+ F\_ATT1\_VAL = ’Contents of First FILE Attribute’,

+ F\_ATT2\_VAL = ’Contents of Second FILE Attribute’,

+ F\_ATT1\_N\_VALUES = 32,

+ F\_ATT2\_N\_VALUES = 33,

+ RI\_ATT1\_VAL = ’Contents of IMAGE’’s First Attribute’,

+ RI\_ATT1\_N\_VALUES = 35,

+ RI\_ATT2\_N\_VALUES = 6)

integer DFACC\_WRITE, DFNT\_INT16, DFNT\_CHAR8

parameter (DFACC\_WRITE = 2,

+ DFNT\_CHAR8 = 4,

+ DFNT\_INT16 = 22)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgscatt, mgsnatt , mgn2ndx,

+ mgselct, mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer status

integer file\_id, gr\_id, ri\_id, ri\_index

integer\*2 ri\_attr\_2(RI\_ATT2\_N\_VALUES)

integer i

do 10 i = 1, RI\_ATT2\_N\_VALUES

ri\_attr\_2(i) = i

10 continue

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Open the HDF file.

C

file\_id = hopen(FILE\_NAME, DFACC\_WRITE, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Set two file attributes to the file with names, data type, numbers of

C values, and values of attributes specified.

C

status = mgscatt(gr\_id, F\_ATT1\_NAME, DFNT\_CHAR8,

+ F\_ATT1\_N\_VALUES, F\_ATT1\_VAL)

status = mgscatt(gr\_id, F\_ATT2\_NAME, DFNT\_CHAR8,

+ F\_ATT2\_N\_VALUES, F\_ATT2\_VAL)

C

C Obtain the index of the image named IMAGE\_NAMR.

C

ri\_index = mgn2ndx(gr\_id, IMAGE\_NAME)

C

C Obtain the identifier of this image.

C

ri\_id = mgselct(gr\_id, ri\_index)

C

C Set two attributes of the image with names, data types, number of

C values, and values of the attributes specified.

C

status = mgscatt(ri\_id, RI\_ATT1\_NAME, DFNT\_CHAR8,

+ RI\_ATT1\_N\_VALUES, RI\_ATT1\_VAL)

status = mgsnatt(ri\_id, RI\_ATT2\_NAME, DFNT\_INT16,

+ RI\_ATT2\_N\_VALUES, ri\_attr\_2)

C

C Terminate access to the image and to the GR interface,

C and close the HDF file.

C

status = mgendac(ri\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

end

### 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 attached to the object - 1. Given a GR interface or raster image identifier and an attribute name, GRfindattr will return a valid attribute index of the file or raster image attribute if the attribute exists. The attribute index can then be used to retrieve information about the attribute or its values. Given a GR interface or raster image identifier and a valid attribute index, GRattrinfo returns the name, data type, and number of values for the file or raster image attribute if the attribute exists.

The syntax for GRfindattr and GRattrinfo is as follows:

C: attr\_index = GRfindattr(obj\_id, attr\_name);

status = GRattrinfo(obj\_id, attr\_index, attr\_name, &data\_type, &n\_values);

FORTRAN: attr\_index = mgfndat(obj\_id, attr\_name)

status = mgatinf(obj\_id, attr\_index, attr\_name, data\_type, n\_values)

The parameter obj\_id is either a GR interface identifier or a raster image identifier. The parameter attr\_name specifies the name of the attribute. The parameter attr\_index specifies the index of the attribute to be read. The attribute index is a zero-based integer and must be less than the total number of attributes assigned to the specified object. The parameter data\_type specifies the data type of the attribute. And the parameter n\_values specifies the number of attribute values.

GRfindattr returns the attribute index if successful and FAIL (or -1) otherwise. GRattrinfo returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. The parameters for GRfindattr and GRattrinfo are further described in Table 8I.

### Reading User-defined Attributes: GRgetattr

GRgetattr reads the values of an attribute assigned to the object identified by the parameter obj\_id. The syntax for GRgetattr is as follows:

C: status = GRgetattr(obj\_id, attr\_index, values);

FORTRAN: status = mggnatt(obj\_id, attr\_index, values)

OR status = mggcatt(obj\_id, attr\_index, values)

The parameter obj\_id is either a GR interface identifier or a raster image identifier. The parameter attr\_index specifies the index of the attribute to be read. The attribute index is a zero-based integer and must be less than the total number of attributes assigned to the specified object.

It is assumed that the buffer values, 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 n\_values\*sizeof(data\_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.

Note that the FORTRAN-77 version of GRgetattr has two routines; mggnatt reads numeric attribute data and mggcatt reads character attribute data.

GRgetattr returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. The parameters for GRgetattr are further described in Table 8I.

* GRsetattr, GRfindattr, GRattrinfo, and GRgetattr Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRsetattr  [intn]  (mgsnatt/mgscatt) | obj\_id | int32 | integer | GR interface or raster image identifier |
| attr\_name | char \* | character\*(\*) | Name assigned to the attribute |
| data\_type | int32 | integer | Data type of the attribute |
| n\_values | int32 | integer | Number of values in the attribute |
| values | VOIDP | <valid numeric data type>(\*)/character\*(\*) | Buffer with the attribute values |
| GRfindattr  [int32]  (mgfndat) | obj\_id | int32 | integer | GR interface or raster image identifier |
| attr\_name | char \* | character\*(\*) | Name of the attribute |
| GRattrinfo  [intn]  (mgatinf) | obj\_id | int32 | integer | GR interface or raster image identifier |
| attr\_index | int32 | integer | Index for the attribute to be read |
| attr\_name | char \* | character\*(\*) | Name of the attribute |
| data\_type | int32 \* | integer | Data type of the attribute values |
| n\_values | int32 \* | integer | Total number of values in the attribute |
| GRgetattr  [intn]  (mggnatt/  mggcatt) | obj\_id | int32 | integer | GR interface or raster image identifier |
| attr\_index | int32 | integer | Index for the attribute to be read |
| values | VOIDP | <valid numeric data type>(\*)/character\*(\*) | Buffer for the attribute values |

* Obtaining File and Image Attributes.

This example illustrates the use of the routines GRattrinfo/mgatinf, GRfindattr/mgfndat, and GRgetattr/mggnatt/mggcatt to extract information and values of file and image attributes that were set by the program in Example 5.

In this example, the program gets the information about each file attribute, then extracts its values. The program then selects the second image in the file, finds the attribute named "Image Attribute 2", obtains the data type and the number of values in the attribute, and extracts its stored values.

C version

C:

#include "hdf.h"

#define FILE\_NAME "General\_RImages.hdf"

#define RI\_ATTR\_NAME "Image Attribute 2"

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 gr\_id, ri\_id, file\_id,

f\_att\_index, /\* index of file attributes \*/

ri\_att\_index, /\* index of raster image attributes \*/

data\_type, /\* image data type \*/

n\_values, /\* number of values in an attribute \*/

value\_index, /\* index of values in an attribute \*/

n\_rimages, /\* number of raster images in the file \*/

n\_file\_attrs; /\* number of file attributes \*/

char attr\_name[MAX\_GR\_NAME]; /\* buffer to hold the attribute name \*/

VOIDP data\_buf; /\* buffer to hold the attribute values \*/

int16 \*int\_ptr; /\* int16 pointer to point to a void data buffer \*/

char8 \*char\_ptr; /\* char8 pointer to point to a void data buffer \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the HDF file.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_READ, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Determine the number of attributes in the file.

\*/

status = GRfileinfo (gr\_id, &n\_rimages, &n\_file\_attrs);

if (status != FAIL && n\_file\_attrs > 0)

{

for (f\_att\_index = 0; f\_att\_index < n\_file\_attrs; f\_att\_index++)

{

/\*

\* Get information about the current file attribute.

\*/

status = GRattrinfo (gr\_id, f\_att\_index, attr\_name, &data\_type,

&n\_values);

/\*

\* Allocate a buffer to hold the file attribute data. In this example,

\* knowledge about the data type is assumed to be available from

\* the previous example for simplicity. In reality, the size

\* of the type must be determined based on the machine where the

\* program resides.

\*/

if (data\_type == DFNT\_CHAR8)

{

data\_buf = malloc (n\_values \* sizeof (char8));

if (data\_buf == NULL)

{

printf ("Unable to allocate space for attribute data.\n");

exit (1);

}

}

else

{

printf ("Unable to determine data type to allocate data buffer.\n");

exit (1);

}

/\*

\* Read and display the attribute values.

\*/

status = GRgetattr (gr\_id, f\_att\_index, (VOIDP)data\_buf);

char\_ptr = (char8 \*) data\_buf;

printf ("Attribute %s: ", attr\_name);

for (value\_index = 0; value\_index < n\_values; value\_index++)

printf ("%c", char\_ptr[value\_index]);

printf ("\n");

/\*

\* Free the space allocated for the data buffer.

\*/

free (data\_buf);

} /\* for \*/

} /\* if \*/

/\*

\* Select the second image in the file.

\*/

ri\_id = GRselect (gr\_id, 1);

/\*

\* Find the image attribute named RI\_ATTR\_NAME.

\*/

ri\_att\_index = GRfindattr (ri\_id, RI\_ATTR\_NAME);

/\*

\* Get information about the attribute.

\*/

status = GRattrinfo (ri\_id, ri\_att\_index, attr\_name, &data\_type, &n\_values);

/\*

\* Allocate a buffer to hold the file attribute data. As mentioned above,

\* knowledge about the data type is assumed to be available from

\* the previous example for simplicity. In reality, the size of the

\* type must be determined based on the machine where the program resides.

\*/

if (data\_type == DFNT\_INT16)

data\_buf = malloc (n\_values \* sizeof (int16));

/\*

\* Read and display the attribute values.

\*/

status = GRgetattr (ri\_id, ri\_att\_index, (VOIDP)data\_buf);

printf ("\nAttribute %s: ", RI\_ATTR\_NAME);

int\_ptr = (int16 \*)data\_buf;

for (value\_index = 0; value\_index < n\_values; value\_index++)

printf ("%d ", int\_ptr[value\_index]);

printf ("\n");

/\*

\* Free the space allocated for the data buffer.

\*/

free (data\_buf);

/\*

\* Terminate access to the raster image and to the GR interface, and

\* close the file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program get\_attribute

implicit none

C

C Parameter declaration

C

character\*19 FILE\_NAME

character\*17 RI\_ATTR\_NAME

C

parameter (FILE\_NAME = ’General\_RImages.hdf’,

+ RI\_ATTR\_NAME = ’Image Attribute 2’)

integer DFACC\_READ, DFNT\_INT16, DFNT\_CHAR8

parameter (DFACC\_READ = 1,

+ DFNT\_CHAR8 = 4,

+ DFNT\_INT16 = 22)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgfinfo, mgatinf, mggcatt, mggnatt , mgfndat,

+ mgselct, mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer status

integer file\_id, gr\_id, ri\_id

integer f\_att\_index, ri\_att\_index, data\_type, n\_values

integer n\_rimages, n\_file\_attrs

integer\*2 int\_buf(10)

character\*17 attr\_name

character\*80 char\_buf

integer i

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Open the HDF file.

C

file\_id = hopen(FILE\_NAME, DFACC\_READ, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Determine the number of attributes in the file.

C

status = mgfinfo(gr\_id, n\_rimages, n\_file\_attrs)

if ((status .NE. -1) .AND. (n\_file\_attrs .GT. 0)) then

do 10 f\_att\_index = 0, n\_file\_attrs-1

C

C Get information about the current file attribute.

C

status = mgatinf(gr\_id, f\_att\_index, attr\_name, data\_type,

+ n\_values)

C

C Check whether data type is DFNT\_CHAR8 in order to use allocated buffer.

C

if(data\_type .NE. DFNT\_CHAR8) then

write(\*,\*)

+ ’Unable to determine data type to use allocated buffer’

else

C

C Read and display the attribute values.

C

status = mggcatt(gr\_id, f\_att\_index, char\_buf)

write(\*,\*) ’Attribute ’, attr\_name, ’ : ’,

+ char\_buf(1:n\_values)

endif

10 continue

endif

C

C Select the second image in the file.

C

ri\_id = mgselct(gr\_id, 1)

C

C Find the image attribute named RI\_ATTR\_NAME.

C

ri\_att\_index = mgfndat(ri\_id, RI\_ATTR\_NAME)

C

C Get information about the attribute.

C

status = mgatinf(ri\_id, ri\_att\_index, attr\_name, data\_type,

+ n\_values)

C

C Read and display attribute values.

C

status = mggnatt(ri\_id, ri\_att\_index, int\_buf)

write(\*,\*) ’Attributes :’, (int\_buf(i), i = 1, n\_values)

C

C Terminate access to the image and to the GR interface,

C and close the HDF file.

C

status = mgendac(ri\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

end

## Reading and Writing Palette Data Using the GR Interface

The GR API library includes routines that read, write, and access information about palette data attached to GR 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 interface routines.

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

### Obtaining a Palette Identifier: GRgetlutid

Given a palette index, the routine GRgetlutid is used to get the palette identifier for the specified palette.

The GRgetlutid function takes two arguments, ri\_id, the raster image identifier of the image that has the palette attached to it, and lut\_index, the index of the palette, and returns the value of the palette identifier corresponding to the specified image. The syntax of GRgetlutid is as follows:

C: pal\_id = GRgetlutid(ri\_id, lut\_index);

FORTRAN: pal\_id = mggltid(ri\_id, lut\_index)

GRgetlutid returns the value of the palette identifier if successful and FAIL (or -1) otherwise. The GRgetlutid parameters are further discussed in Table 8J.

### Obtaining the Number of Palettes Associated with an Image: GRgetnluts

Given an image identifier, GRgetnluts is used to determne the number of palettes currently associated with an image.

The GRgetnluts function takes one argument, ri\_id, a raster image identifier, and returns the number of palettes associated with that imare. The syntax of GRgetnluts is as follows:

C: n\_luts = GRgetnluts(ri\_id);

FORTRAN: n\_luts = mggnluts(ri\_id)

GRgetnluts returns the number of palettes associated with the identified image if successful and FAIL (or -1) otherwise. The GRgetnluts parameters are further discussed in Table 8J.

### Obtaining the Reference Number of a Specified Palette: GRluttoref

Given a palette identifier, GRluttoref can be used to obtain the reference number of the specified palette.

The GRluttoref routine takes one argument, pal\_id, a palette identifier, and returns the reference number of the palette. GRluttoref is commonly used to annotate the palette or to include the palette within a vgroup. The syntax of GRgetlutid is as follows:

C: pal\_ref = GRluttoref(pal\_id);

FORTRAN: pal\_ref = mglt2rf(pal\_id)

GRluttoref returns the reference number of the palette if successful and 0 otherwise. The GRluttoref parameters are further discussed in Table 8J.

* GRgetlutid, GRgetlutinfo, and GRluttoref Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRgetlutid  [int32]  (mggltid) | ri\_id | int32 | integer | Raster image identifier |
| lut\_index | int32 | integer | Palette index |
| GRluttoref  [uint16]  (mglt2rf) | pal\_id | int32 | integer | Palette identifier |
| GRgetnluts  [intn]  (mggnluts) | ri\_id | int32 | integer | Raster image identifier |

### Obtaining Palette Information: GRgetlutinfo

Given a palette identifier, GRgetlutinfo retrieves information about the palette and its components.

The GRgetlutinfo function takes one input argument, pal\_id, the identifier of the palette, and several return parameters. The return parameters are n\_comps, the number of components of the palette; data\_type, the data type of the palette data; interlace\_mode, the interlace mode of the stored palette data; and num\_entries, the number of entries in the palette. The syntax of GRgetlutinfo is as follows:

C: status = GRgetlutinfo(pal\_id, &n\_comps, &data\_type, &interlace\_mode, &num\_entries);

FORTRAN: status = mgglinf(pal\_id, n\_comps, data\_type, interlace\_mode, num\_entries)

GRgetlutinfo returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. The GRgetlutinfo parameters are further discussed in Table 8J.

### Writing Palette Data: GRwritelut

GRwritelut writes palette data into the palette identified by the parameter pal\_id. The syntax of GRwritelut is as follows:

C: status = GRwritelut(pal\_id, n\_comps, data\_type, interlace\_mode, num\_entries, pal\_data);

FORTRAN: status = mgwrlut(pal\_id, n\_comps, data\_type, interlace\_mode, num\_entries, pal\_data)

OR status = mgwclut(pal\_id, n\_comps, data\_type, interlace\_mode, num\_entries, pal\_data)

The parameter n\_comps specifies the number of pixel components in the palette; it must have a value of at least 1. The parameter data\_type specifies the data type of the palette data. Refer to (See Table 2F on page 14) for all data types supported by HDF.

The parameter interlace\_mode specifies the interlacing in which the palette is to be written. The valid values of interlace\_mode are: MFGR\_INTERLACE\_PIXEL (or 0), MFGR\_INTERLACE\_LINE (or 1) and MFGR\_INTERLACE\_COMPONENT (or 2). Refer to Section 8.5.1 on page 300 for further information.

The parameter num\_entries specifies the number of entries in the palette. The buffer pal\_data contains the palette data.

Note that the FORTRAN-77 version of GRwritelut has two routines; mgwrlut writes buffered numeric palette data and mgwclut writes buffered character palette data.

GRwritelut returns either SUCCEED (or 0) or FAIL (or -1). The parameters of this routine are further defined in (See Table 8K on page 342).

### Setting the Interlace Mode for a Palette: GRreqlutil

GRreqlutil sets the interlace mode for the next palette to be read. The syntax of GRreqlutil is as follows:

C: status = GRreqlutil(pal\_id, interlace\_mode);

FORTRAN: status = mgrltil(pal\_id, interlace\_mode)

The parameter interlace\_mode specifies the interlacing that will be in effect for the next palette read operation. The valid values of interlace\_mode are: MFGR\_INTERLACE\_PIXEL (or 0), MFGR\_INTERLACE\_LINE (or 1) and MFGR\_INTERLACE\_COMPONENT (or 2). Refer to Section 8.5.1 on page 300 for further information.

GRreqlutil may be called at anytime before the read operation of the specified palette. In addition, it may be called more than once; the interlace mode setting specified by the last call to the routine will be used for the next read operation.

GRreqlutil returns either SUCCEED (or 0) or FAIL (or -1). The parameters of this routine are further defined in (See Table 8K on page 342).

### Reading Palette Data: GRreadlut

GRreadlut reads data from the palette identified by the parameter pal\_id. The syntax of GRreadlut is as follows:

C: status = GRreadlut(pal\_id, pal\_data);

FORTRAN: status = mgrdlut(pal\_id, pal\_data)

OR status = mgrclut(pal\_id, pal\_data)

The read data will be stored in the buffer pal\_data, which is assumed to be sufficient to store the read palette data. The sufficient amount of space needed can be determined using the routine GRgetlutinfo. The palette data is read according to the interlacing mode set by the last call to GRreqlutil.

Note that the FORTRAN-77 version of GRreadlut has two routines; mgrdlut reads numeric palette data and mgrclut reads character palette data.

GRreadlut returns either SUCCEED (or 0) or FAIL (or -1). The parameters of this routine are further defined in Table 8K.

* GRgetlutid, GRwritelut, GRreqlutil, and GRreadlut Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRgetlutinfo  [intn]  (mgglinf) | pal\_id | int32 | integer | Palette identifier |
| n\_comps | int32\* | integer | Number of components in each palette element |
| data\_type | int32\* | integer | Data type of the palette data |
| interlace\_mode | int32\* | integer | Interlace mode of the palette data |
| num\_entries | int32\* | integer | Buffer for the size of the palette |
| GRwritelut  [intn]  (mgwrlut/  mgwclut) | pal\_id | int32 | integer | Palette identifier |
| n\_comps | int32 | integer | Number of components in each palette element |
| data\_type | int32 | integer | Type of the palette data |
| interlace\_mode | int32 | integer | Interlace mode of the palette data |
| num\_entries | int32 | integer | Number of entries in the palette |
| pal\_data | VOIDP | <valid numeric data type>(\*)/character\*(\*) | Buffer for the palette data to be written |
| GRreqlutil  [intn]  (mgrltil) | pal\_id | int32 | integer | Palette identifier |
| interlace\_mode | intn | integer | Interlace mode for the next palette read operation |
| GRreadlut  [intn]  (mgrdlut/  mgrclut) | pal\_id | int32 | integer | Palette identifier |
| pal\_data | VOIDP | <valid numeric data type>(\*)/character\*(\*) | Buffer for the palette data to be read |

* Writing a Palette.

This example illustrates the use of the routines GRgetlutid/mggltid and GRwritelut/mgwclut to attach a palette to a raster image and write data to it.

In this example, the program creates an image named "Image with Palette" in the file "Image\_with\_Palette.hdf". A palette is then attached to the image and data is written to it.

C version

C:

#include "hdf.h"

#define FILE\_NAME "Image\_with\_Palette.hdf"

#define NEW\_IMAGE\_NAME "Image with Palette"

#define N\_COMPS\_IMG 2 /\* number of image components \*/

#define X\_LENGTH 5

#define Y\_LENGTH 5

#define N\_ENTRIES 256 /\* number of entries in the palette \*/

#define N\_COMPS\_PAL 3 /\* number of palette’s components \*/

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status, /\* status for functions returning an intn \*/

i, j;

int32 file\_id, gr\_id, ri\_id, pal\_id,

interlace\_mode,

start[2], /\* holds where to start to write for each dimension \*/

edges[2], /\* holds how long to write for each dimension \*/

dim\_sizes[2]; /\* sizes of the two dimensions of the image array \*/

uint8 image\_buf[Y\_LENGTH][X\_LENGTH][N\_COMPS\_IMG]; /\* data of first image \*/

uint8 palette\_buf[N\_ENTRIES][N\_COMPS\_PAL];

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the HDF file.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_CREATE, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Define the dimensions and interlace mode of the image.

\*/

dim\_sizes[0] = X\_LENGTH;

dim\_sizes[1] = Y\_LENGTH;

interlace\_mode = MFGR\_INTERLACE\_PIXEL;

/\*

\* Create the image named NEW\_IMAGE\_NAME.

\*/

ri\_id = GRcreate (gr\_id, NEW\_IMAGE\_NAME, N\_COMPS\_IMG, DFNT\_UINT8,

interlace\_mode, dim\_sizes);

/\*

\* Fill the image data buffer with values.

\*/

for (i = 0; i < Y\_LENGTH; i++)

{

for (j = 0; j < X\_LENGTH; j++)

{

image\_buf[i][j][0] = (i + j) + 1;

image\_buf[i][j][1] = (i + j) + 2;

}

}

/\*

\* Define the size of the data to be written, i.e., start from the origin

\* and go as long as the length of each dimension.

\*/

start[0] = start[1] = 0;

edges[0] = X\_LENGTH;

edges[1] = Y\_LENGTH;

/\*

\* Write the data in the buffer into the image array.

\*/

status = GRwriteimage (ri\_id, start, NULL, edges, (VOIDP)image\_buf);

/\*

\* Initialize the palette to grayscale.

\*/

for (i = 0; i < N\_ENTRIES; i++) {

palette\_buf[i][0] = i;

palette\_buf[i][1] = i;

palette\_buf[i][2] = i;

}

/\*

\* Define palette interlace mode.

\*/

interlace\_mode = MFGR\_INTERLACE\_PIXEL;

/\*

\* Get the identifier of the palette attached to the image NEW\_IMAGE\_NAME.

\*/

pal\_id = GRgetlutid (ri\_id, 0);

/\*

\* Write data to the palette.

\*/

status = GRwritelut (pal\_id, N\_COMPS\_PAL, DFNT\_UINT8, interlace\_mode,

N\_ENTRIES, (VOIDP)palette\_buf);

/\*

\* Terminate access to the image and to the GR interface, and

\* close the HDF file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program write\_palette

implicit none

C

C Parameter declaration

C

character\*22 FILE\_NAME

character\*18 NEW\_IMAGE\_NAME

integer X\_LENGTH

integer Y\_LENGTH

integer N\_ENTRIES

integer N\_COMPS\_IMG

integer N\_COMPS\_PAL

C

parameter (FILE\_NAME = ’Image\_with\_Palette.hdf’,

+ NEW\_IMAGE\_NAME = ’Image with Palette’,

+ X\_LENGTH = 5,

+ Y\_LENGTH = 5,

+ N\_ENTRIES = 256,

+ N\_COMPS\_IMG = 2,

+ N\_COMPS\_PAL = 3)

integer DFACC\_CREATE, DFNT\_CHAR8, DFNT\_UINT8, MFGR\_INTERLACE\_PIXEL

parameter (DFACC\_CREATE = 4,

+ DFNT\_CHAR8 = 4,

+ DFNT\_UINT8 = 21,

+ MFGR\_INTERLACE\_PIXEL = 0)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgcreat, mgwcimg, mggltid, mgwclut,

+ mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer file\_id, gr\_id, ri\_id, pal\_id

integer interlace\_mode

integer start(2), stride(2), edges(2), dim\_sizes(2)

integer status

integer i, j

character image\_buf(N\_COMPS\_IMG, X\_LENGTH, Y\_LENGTH)

character palette\_buf(N\_COMPS\_PAL, N\_ENTRIES)

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Create and open the file.

C

file\_id = hopen(FILE\_NAME, DFACC\_CREATE, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Define interlace mode and dimensions of the image.

C

interlace\_mode = MFGR\_INTERLACE\_PIXEL

dim\_sizes(1) = X\_LENGTH

dim\_sizes(2) = Y\_lENGTH

C

C Create the raster image array.

C

ri\_id = mgcreat(gr\_id, NEW\_IMAGE\_NAME, N\_COMPS\_IMG, DFNT\_CHAR8,

+ interlace\_mode, dim\_sizes)

C

C Fill the image data buffer with values.

C

do 20 i = 1, Y\_LENGTH

do 10 j = 1, X\_LENGTH

image\_buf(1,j,i) = char(i + j - 1 )

image\_buf(2,j,i) = char(i + j)

10 continue

20 continue

C

C Define the size of the data to be written, i.e., start from the origin

C and go as long as the length of each dimension.

C

start(1) = 0

start(2) = 0

edges(1) = X\_LENGTH

edges(2) = Y\_LENGTH

stride(1) = 1

stride(2) = 1

C

C Write the data in the buffer into the image array.

C

status = mgwcimg(ri\_id, start, stride, edges, image\_buf)

C

C Initilaize the palette buffer to grayscale.

C

do 40 i = 1, N\_ENTRIES

do 30 j = 1, N\_COMPS\_PAL

palette\_buf(j,i) = char(i)

30 continue

40 continue

C

C Get the identifier of the palette attached to the image NEW\_IMAGE\_NAME.

C

pal\_id = mggltid(ri\_id, 0)

C

C Set palette interlace mode.

C

interlace\_mode = MFGR\_INTERLACE\_PIXEL

C

C Write data to the palette.

C

status = mgwclut(pal\_id, N\_COMPS\_PAL, DFNT\_UINT8, interlace\_mode,

+ N\_ENTRIES, palette\_buf)

C

C Terminate access to the raster image and to the GR interface,

C and close the HDF file.

C

status = mgendac(ri\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

end

* Reading a Palette.

This example illustrates the use of the routines GRgetlutinfo/mgglinf and GRreadlut/mgrclut to obtain information about a palette and to read palette data.

In this example, the program finds and selects the image named "Image with Palette" in the file "Image\_with\_Palette.hdf". Then the program obtains information about the palette and reads the palette data.

C version

C:

#include "hdf.h"

#define FILE\_NAME "Image\_with\_Palette.hdf"

#define IMAGE\_NAME "Image with Palette"

#define N\_ENTRIES 256 /\* number of elements of each color \*/

main( )

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status, /\* status for functions returning an intn \*/

i, j;

int32 file\_id, gr\_id, ri\_id, pal\_id, ri\_index;

int32 data\_type, n\_comps, n\_entries, interlace\_mode;

uint8 palette\_data[N\_ENTRIES][3]; /\* static because of fixed size \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the file.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_READ, 0);

/\*

\* Initiate the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Get the index of the image IMAGR\_NAME.

\*/

ri\_index = GRnametoindex (gr\_id, IMAGE\_NAME);

/\*

\* Get image identifier.

\*/

ri\_id = GRselect (gr\_id, ri\_index);

/\*

\* Get the identifier of the palette attached to the image.

\*/

pal\_id = GRgetlutid (ri\_id, ri\_index);

/\*

\* Obtain and display information about the palette.

\*/

status = GRgetlutinfo (pal\_id, &n\_comps, &data\_type, &interlace\_mode,

&n\_entries);

printf ("Palette: %d components; %d entries\n", n\_comps, n\_entries);

/\*

\* Read the palette data.

\*/

status = GRreadlut (pal\_id, (VOIDP)palette\_data);

/\*

\* Display the palette data. Recall that HDF supports only 256 colors.

\* Each color is defined by its 3 components. Therefore,

\* verifying the value of n\_entries and n\_comps is not necessary and

\* the buffer to hold the palette data can be static. However,

\* if more values or colors are added to the model, these parameters

\* must be checked to allocate sufficient space when reading a palette.

\*/

printf (" Palette Data: \n");

for (i=0; i< n\_entries; i++)

{

for (j = 0; j < n\_comps; j++)

printf ("%i ", palette\_data[i][j]);

printf ("\n");

}

printf ("\n");

/\*

\* Terminate access to the image and to the GR interface, and

\* close the HDF file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

FORTRAN-77 version

FORTRAN:

program read\_palette

implicit none

C

C Parameter declaration

C

character\*22 FILE\_NAME

character\*18 IMAGE\_NAME

integer N\_ENTRIES

integer N\_COMPS\_PAL

C

parameter (FILE\_NAME = ’Image\_with\_Palette.hdf’,

+ IMAGE\_NAME = ’Image with Palette’,

+ N\_COMPS\_PAL = 3,

+ N\_ENTRIES = 256)

integer DFACC\_READ, DFNT\_CHAR8, DFNT\_UINT8, MFGR\_INTERLACE\_PIXEL

parameter (DFACC\_READ = 1,

+ DFNT\_CHAR8 = 4,

+ DFNT\_UINT8 = 21,

+ MFGR\_INTERLACE\_PIXEL = 0)

C

C Function declaration

C

integer hopen, hclose

integer mgstart, mgn2ndx, mgselct, mggltid, mgglinf,

+ mgrclut, mgendac, mgend

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer file\_id, gr\_id, ri\_id, ri\_index, pal\_id, pal\_index

integer interlace\_mode

integer data\_type, n\_comps, n\_entries\_out

integer status

integer i, j

character palette\_data(N\_COMPS\_PAL, N\_ENTRIES)

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Open the file.

C

file\_id = hopen(FILE\_NAME, DFACC\_READ, 0)

C

C Initialize the GR interface.

C

gr\_id = mgstart(file\_id)

C

C Get the index of the image IMAGE\_NAME.

C

ri\_index = mgn2ndx(gr\_id, IMAGE\_NAME)

C

C Get the image identifier.

C

ri\_id = mgselct(gr\_id, 0)

C

C Get the identifier of the palette attached to the image.

C

pal\_index = 0

pal\_id = mggltid(ri\_id, pal\_index)

C

C Obtain information about the palette.

C

status = mgglinf(pal\_id, n\_comps, data\_type, interlace\_mode,

+ n\_entries\_out)

write(\*,\*) ’ Palette: ’, n\_comps, ’ components; ’,

+ n\_entries\_out, ’ entries’

C

C Read the palette.

C

status = mgrclut(pal\_id, palette\_data)

C

C Display the palette data.

C

write(\*,\*) "Palette data"

do 10 i = 1, n\_entries\_out

write(\*,\*) (ichar(palette\_data(j,i)), j = 1, n\_comps)

10 continue

C

C Terminate access to the raster image and to the GR interface,

C and close the HDF file.

C

status = mgendac(ri\_id)

status = mgend(gr\_id)

status = hclose(file\_id)

end

## Chunked Raster Images

The GR interface also supports chunking in a manner similar to that of the SD interface. There is one restriction on a raster image: it must be created with MFGR\_INTERLACE\_PIXEL (or 0) in the call to GRcreate. We refer the reader to Section 3.11 of Chapter 3, Scientific Data Sets (SD API), and to Chapter 14, HDF Performance Issues, for discussions of chunking concepts and performance related topics. The GR interface provides three routines, GRsetchunk, GRsetchunkcache, and GRgetchunkinfo, to create and maintain chunked raster images. The generic functions for reading and writing GR images, GRwriteimage and GRreadimage, will write and read chunked raster images as well. However, the GR interface provides special write and read routines, GRwritechunk and GRreadchunk, which are similar to SDwritechunk and SDreadchunk. Compared toGRwriteimage and GRreadimage, GRwritechunk and GRreadchunk are low-overhead but are only sutable for writing or reading complete chunks.

### Difference between a Chunked Raster Image and a Chunked SDS

Chunks of scientific datasets (SDSs) have the same dimensionality as the SDS itself and the chunks can divide the SDS along any dimension. While raster images under the GR interface are actually 3-dimensional arrays, 2 dimensions define the image while the third dimension (the stack of 2-dimensional image planes) provides the composite definition of the color at each pixel of the 2-dimensional image. Chunking can be applied only across the 2-dimensions of the image; chunking cannot divide the array across the third dimension. In other words, all of the elements of the raster image that define a single pixel must remain together in the same chunk.

* Chunks in a GR raster image dataset



### Making a Raster Image a Chunked Raster Image: GRsetchunk

GRsetchunk makes the raster image, identified by the parameter ri\_id, a chunked raster image according to the provided chunking and compression information. The syntax of GRsetchunk is as follows:

C: status = GRsetchunk(ri\_id, c\_def, flags);

FORTRAN: status = mgschnk(ri\_id, dim\_length, comp\_type, comp\_prm)

The parameters c\_def and flags in C or the parameters comp\_type and comp\_prm in FORTRAN-77 provide the chunking and compression information and are discussed below.

In C:

The parameter c\_def is a union of type HDF\_CHUNK\_DEF, which is defined as follows:

* typedef union hdf\_chunk\_def\_u
* {
* int32 chunk\_lengths[2]; /\* chunk lengths along each dim \*/
* struct
* {
* int32 chunk\_lengths[2];
* int32 comp\_type; /\* compression type \*/
* struct comp\_info cinfo;
* } comp;
* struct
* {
* /\* is not used in GR interface \*/
* } nbit;
* } HDF\_CHUNK\_DEF

Valid values of the parameter flags are HDF\_CHUNK for chunked and uncompressed data and (HDF\_CHUNK | HDF\_COMP) for chunked and compressed data. Data can be compressed using run-length encoding (RLE), Skipping Huffman, GZIP, or Szip compression algorithms.

If the parameter flags has a value of HDF\_CHUNK, the chunk dimensions must be specified in the field c\_def.chunk\_lengths[]. If the parameter flags has a value of (HDF\_CHUNK | HDF\_COMP), the chunk dimensions must be specified in the field c\_def.comp.chunk\_lengths[] and the compression type in the field c\_def.comp.comp\_type. Valid values of compression type values are:

COMP\_CODE\_NONE (or 0) for uncompressed data

COMP\_CODE\_RLE (or 1) for RLE compression

COMP\_CODE\_SKPHUFF (or 3) for Skipping Huffman compression

COMP\_CODE\_DEFLATE (or 4) for GZIP compression

COMP\_CODE\_SZIP (or 5) for Szip compression

For Skipping Huffman, GZIP, and Szip compression methods, parameters are passed in corresponding fields of the structure cinfo. Specify skipping size for Skipping Huffman compression in the field c\_def.comp.cinfo.skphuff.skp\_size; this value cannot be less than 1. Specify deflate level for GZIP compression in the field c\_def.comp.cinfo.deflate\_level. Valid values of deflate levels are integers from 0 to 9 inclusive. Specify the Szip options mask and the number of pixels per block in a chunked and Szip-compressed dataset in the fields c\_info.szip.options\_mask and c\_info.szip.pixels\_per\_block, respectively.

Refer to the discussion of SDsetcompress routine in Section 3.5.2 on page 47for the definition of the structure comp\_info.

In FORTRAN-77:

Chunk dimensions are specified in the array dim\_length and the compression type in the parameter comp\_type. Valid compression types and their values are defined in the hdf.inc file and are listed below:

COMP\_CODE\_NONE (or 0) for uncompressed data

COMP\_CODE\_RLE (or 1) for RLE compression

COMP\_CODE\_SKPHUFF (or 3) for Skipping Huffman compression

COMP\_CODE\_DEFLATE (or 4) for GZIP compression

The parameter comp\_prm specifies the compression parameters for the Skipping Huffman and GZIP compression methods. It contains only one element which is set to the skipping size for Skipping Huffman compression or the deflate level for GZIP compression. Currently, Szip compression is not yet supported by Fortran GR interface.

GRsetchunk returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. The GRsetchunk parameters are discussed further in (See Table 8L on page 361)

### Writing a Chunked Raster Image: GRwritechunk

GRwritechunk is used to write a chunk of a chunked raster image. The syntax of the GRwritechunk routine is as follows:

C: status = GRwritechunk(ri\_id, &origin, &datap);

FORTRAN: status = mgwchnk(ri\_id, origin, datap)

status = mgwcchnk(ri\_id, origin, datap)

GRwritechunk writes the entire chunk of data stored in the buffer datap to the chunked raster image identified by the parameter ri\_id. Writing starts at the location specified by the parameter origin. This function has less overhead than GRwriteimage and should be used whenever an entire chunk of data is to be written.

The raster image must be stored in pixel-interlace mode.

The parameter origin is a two-dimensional array which specifies the coordinates of the chunk according to the chunk position in the overall chunk array.

The datap buffer contains the chunk data. The data must be organized in pixel-interlace mode.

Note that the FORTRAN-77 version of GRwritechunk has two routines; mgwchnk writes buffered numeric data and mgwcchnk writes buffered character data.

GRwritechunk returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. The GRwritechunk parameters are discussed further in Table 8L.

* Creating and Writing a Chunked Raster Image

This example illustrates the use of the routines Hopen/hopen, GRstart/mgstart, GRcreate/mgcreat, GRwritechunk/mgwchnk, GRendaccess/mgendac, GRend/mgend, and Hclose/hclose to create an HDF file and store a raster image in it.

In this example, the program creates an image of 6 rows by 10 columns in C and 10 rows by 6 columns in FORTRAN. The image is set up to be chunked with a chunk size of 3x2 in C and 2x3 in FORTRAN and compressed with the GZIP method. Three chunks are then written to the image. See Figure 8c through Figure 8d for illustrations.

* Chunked GR image as written by C example



* Chunked GR image as written by FORTRAN example



C:

#include "hdf.h"

#define FILE\_NAME “Image\_Chunked.hdf”

#define IMAGE\_NAME “Image with Chunks”

#define X\_LENGTH 10 /\* number of rows in the image \*/

#define Y\_LENGTH 6 /\* number of columns in the image \*/

#define NCOMPS 3 /\* number of components in the image \*/

int main()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 file\_id, /\* HDF file identifier \*/

gr\_id, /\* GR interface identifier \*/

ri\_id, /\* raster image identifier \*/

dims[2], /\* dimension sizes of the image array \*/

origin[2], /\* origin position to write each chunk \*/

interlace\_mode; /\* interlace mode of the image \*/

HDF\_CHUNK\_DEF chunk\_def; /\* Chunk defintion set \*/

int32 chunk00[] = {1, 2, 3, 4, 5, 6,

7, 8, 9, 10, 11, 12,

13, 14, 15, 16, 17, 18 };

int32 chunk01[] = {210, 211, 212, 220, 221, 222,

230, 231, 232, 240, 241, 242,

250, 251, 252, 260, 261, 262};

int32 chunk14[] = {1010, 1011, 1012, 1020, 1021, 1022,

1030, 1031, 1032, 1040, 1041, 1042,

1050, 1051, 1052, 1060, 1061, 1062};

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Create and open the file.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_CREATE, 0);

/\*

\* Initialize the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Set dimensions of the image.

\*/

dims[0] = Y\_LENGTH;

dims[1] = X\_LENGTH;

/\*

\* Create the raster image array.

\*/

ri\_id = GRcreate (gr\_id, IMAGE\_NAME, NCOMPS, DFNT\_INT32,

MFGR\_INTERLACE\_PIXEL, dims);

/\*

\* Define chunked image.

\*/

chunk\_def.comp.comp\_type = COMP\_CODE\_DEFLATE;

chunk\_def.comp.cinfo.deflate.level = 6;

chunk\_def.comp.chunk\_lengths[0] = 3;

chunk\_def.comp.chunk\_lengths[1] = 2;

status = GRsetchunk (ri\_id, chunk\_def, HDF\_CHUNK | HDF\_COMP);

/\*

\* Write first chunk(0,0).

\*/

origin[0] = 0;

origin[1] = 0;

status = GRwritechunk (ri\_id, origin, (VOIDP)chunk00);

/\*

\* Write second chunk(0,1).

\*/

origin[0] = 0;

origin[1] = 1;

status = GRwritechunk (ri\_id, origin, (VOIDP)chunk01);

/\*

\* Write third chunk(1,4).

\*/

origin[0] = 1;

origin[1] = 4;

status = GRwritechunk (ri\_id, origin, (VOIDP)chunk14);

/\*

\* Terminate access to the raster image and to the GR interface and,

\* close the HDF file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

return 0;

}

FORTRAN:

program gr\_chunking\_example

implicit none

C

C Parameter declaraction

C

character\*14 FILE\_NAME

character\*14 DATASET\_NAME

parameter (FILE\_NAME = ’gr\_chunked.hdf’,

. DATASET\_NAME = ’gzip\_comp\_data’)

integer NCOMP, MFGR\_INTERLACE\_PIXEL

parameter(NCOMP = 3, MFGR\_INTERLACE\_PIXEL = 0)

integer DFACC\_CREATE, DFACC\_READ, DFACC\_WRITE

parameter (DFACC\_CREATE = 4,

. DFACC\_READ = 1,

. DFACC\_WRITE = 2)

integer DFNT\_INT32

parameter (DFNT\_INT32 = 24)

integer X\_LENGTH, Y\_LENGTH, X\_CH\_LENGTH, Y\_CH\_LENGTH

parameter (X\_LENGTH = 6,

. Y\_LENGTH = 10,

. X\_CH\_LENGTH = 3,

. Y\_CH\_LENGTH = 2)

C

C Compression parameters.

C

integer COMP\_CODE\_DEFLATE, DEFLATE\_LEVEL

parameter( COMP\_CODE\_DEFLATE = 4, DEFLATE\_LEVEL = 6)

C

C Function declaration.

C

integer mgstart, mgcreat, mgendac, mgend

integer mgwchnk, mgschnk

integer hopen, hclose

C

C\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

integer ri\_id, gr\_id, file\_id

integer dims(2), start(2)

integer status, il

integer comp\_prm(1), comp\_type

C

C Data buffers.

C

integer\*4 chunk11(NCOMP\* X\_CH\_LENGTH\*Y\_CH\_LENGTH)

integer\*4 chunk21(NCOMP\* X\_CH\_LENGTH\*Y\_CH\_LENGTH)

integer\*4 chunk52(NCOMP\* X\_CH\_LENGTH\*Y\_CH\_LENGTH)

C

C Chunking dimension arrays

C

integer ch\_dims(2)

C

C\*\*\*\* End of variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

C

C

C Data initialization

C

data chunk11 / 110, 111, 112, 120, 121, 122,

. 130, 131, 132, 140, 141, 142,

. 150, 151, 152, 160, 161, 162

. /,

. chunk21 /

. 210, 211, 212, 220, 221, 222,

. 230, 231, 232, 240, 241, 242,

. 250, 251, 252, 260, 261, 262

. /,

. chunk52 /

. 1010, 1011, 1012, 1020, 1021, 1022,

. 1030, 1031, 1032, 1040, 1041, 1042,

. 1050, 1051, 1052, 1060, 1061, 1062

. /

C

C Define chunk dimensions.

C

ch\_dims(1) = Y\_CH\_LENGTH

ch\_dims(2) = X\_CH\_LENGTH

C

C Create and open the file and initiate GR interface..

C

file\_id = hopen(FILE\_NAME, DFACC\_CREATE, 0)

gr\_id = mgstart(file\_id)

C

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

C

il = MFGR\_INTERLACE\_PIXEL

dims(1) = X\_LENGTH

dims(2) = Y\_LENGTH

C

C Create GR dataset.

C

ri\_id = mgcreat(gr\_id, DATASET\_NAME, NCOMP, DFNT\_INT32, il, dims)

C

C Define chunked GR dataset using GZIP compression.

C

comp\_prm(1) = DEFLATE\_LEVEL

comp\_type = COMP\_CODE\_DEFLATE

status = mgschnk (ri\_id, ch\_dims, comp\_type, comp\_prm)

C

C Define the location of the first chunk and write the data.

C

start(1) = 1

start(2) = 1

status = mgwchnk(ri\_id, start, chunk11)

C

C Define the location of the second chunk and write the data.

C

start(1) = 2

start(2) = 1

status = mgwchnk(ri\_id, start, chunk21)

C

C Define the location of the third and write the data.

C

start(1) = 5

start(2) = 2

status = mgwchnk(ri\_id, start, chunk52)

C

C Terminate access to the array.

C

status = mgendac(ri\_id)

C

C Terminate access to the GR interface.

C

status = mgend(gr\_id)

C

C Close the file.

C

status = hclose(file\_id)

end

### Reading a Chunked Raster Image: GRreadchunk

GRreadchunk is used to read an entire chunk of data from a chunked raster image. The syntax of the GRreadchunk routine is as follows:

C: status = GRreadchunk(ri\_id, &origin, datap);

FORTRAN: status = mgrchnk(ri\_id, origin, datap)

status = mgrcchnk(ri\_id, origin, datap)

GRreadchunk reads the entire chunk of data stored from the chunked raster image identified by the parameter ri\_id and stores it in the buffer datap. The chunk to be read is specified by the parameter origin. This function has less overhead than GRreadimage and should be used whenever an entire chunk of data is to be read.

The raster image must be stored in pixel-interlace mode.

The parameter origin is a two-dimensional array which specifies the coordinates of the chunk according to the chunk position in the overall chunk array.

The datap buffer contains the chunk data. The data is organized in pixel-interlace mode.

Note that the FORTRAN-77 version of GRreadchunk has two routines; mgrchnk reads numeric data and mgrcchnk reads character data to the buffer.

GRreadchunk returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. GRreadchunk will return FAIL (or -1) when an attempt is made to read from a non-chunked image. The GRreadchunk parameters are discussed further in Table 8L.

* Reading a Chunked Raster Image.

This example illustrates the use of the routines **GRreadchunk/mgrchnk** to read the raster image’s chunked data.

In this example, the program finds and selects the image named "Image with Chunks" in the file "Image\_Chunked.hdf". Then the program obtains information about the image and reads the image data. Only C example is available at this time.

C:

#include "hdf.h"

#define FILE\_NAME "Image\_Chunked.hdf"

#define IMAGE\_NAME "Image with Chunks"

#define X\_LENGTH 10 /\* number of rows in the image \*/

#define Y\_LENGTH 6 /\* number of columns in the image \*/

#define NCOMPS 3 /\* number of components in the image \*/

int main()

{

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

intn status; /\* status for functions returning an intn \*/

int32 file\_id, /\* HDF file identifier \*/

gr\_id, /\* GR interface identifier \*/

ri\_id, /\* raster image identifier \*/

dims[2], /\* dimension sizes of the image array \*/

origin[2], /\* origin position to write each chunk \*/

interlace\_mode; /\* interlace mode of the image \*/

HDF\_CHUNK\_DEF chunk\_def; /\* Chunk defintion set \*/

/\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* Variable declaration \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*/

/\*

\* Open the file.

\*/

file\_id = Hopen (FILE\_NAME, DFACC\_READ, 0);

/\*

\* Initiate the GR interface.

\*/

gr\_id = GRstart (file\_id);

/\*

\* Get the index of the image IMAGR\_NAME.

\*/

ri\_index = GRnametoindex (gr\_id, IMAGE\_NAME);

/\*

\* Get image identifier.

\*/

ri\_id = GRselect (gr\_id, ri\_index);

/\*

\* Set dimensions of the image.

\*/

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 and to the GR interface, and

\* close the HDF file.

\*/

status = GRendaccess (ri\_id);

status = GRend (gr\_id);

status = Hclose (file\_id);

}

### Obtaining Information about a Chunked Raster Image: GRgetchunkinfo

GRgetchunkinfo is used to determine whether a raster image is chunked and how chunking is defined. The syntax of the GRgetchunkinfo routine is as follows:

C: status = GRgetchunkinfo(ri\_id, &c\_def, &flag);

FORTRAN: status = mggichnk(ri\_id, dim\_length, flag)

GRgetchunkinfo retrieves chunking information about the raster image into the parameters c\_def and flag in C and into the parameters dim\_length and flag in FORTRAN-77. Note that only chunk dimensions are retrieved; compression information is not available.

The value returned in the parameter flag indicates whether the raster image is not chunked, chunked, or chunked and compressed. HDF\_NONE (or -1) indicates that the raster image is not chunked. HDF\_CHUNK (or 0) indicates that the raster image is chunked and not compressed. (HDF\_CHUNK | HDF\_COMP) (or 1) indicates that raster image is chunked and compressed with one of the allowed compression methods: RLE, Skipping Huffman, or GZIP.

In C, if the raster image is chunked and not compressed, GRgetchunkinfo fills the array chunk\_lengths in the union c\_def with the values of the corresponding chunk dimensions. If the raster image is chunked and compressed, GRgetchunkinfo fills the array chunk\_lengths in the structure comp of the union c\_def with the values of the corresponding chunk dimensions. Refer to Section 8.12.2 on page 350 on GRsetchunk for specific information on the union HDF\_CHUNK\_DEF. In C, if the chunk length for each dimension is not needed, NULL can be passed in as the value of the parameter c\_def.

In FORTRAN-77, chunk dimensions are retrieved into the array dim\_length.

GRgetchunkinfo returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. The GRgetchunkinfo parameters are discussed further in Table 8L.

### Setting the Maximum Number of Chunks in the Cache: GRsetchunkcache

GRsetchunkcache sets the maximum number of chunks to be cached for chunked raster image. GRsetchunkcache has similar behavior to SDsetchunkcache. Refer to Section 3.12.2 on page 116 for specific information. The syntax of GRsetchunkcache is as follows:

C: status = GRsetchunkcache(ri\_id, maxcache, flags);

FORTRAN: status = mgscchnk(ri\_id, maxcache, flags)

The maximum number of chunks is specified by the parameter maxcache. Currently, the only valid value of the parameter flags is 0.

If GRsetchunkcache is not called, the maximum number of chunks in the cache is set to the number of chunks along the fastest-changing dimension. Since GRsetchunkcache is similar to the routine SDsetchunkcache, refer to Section 3.12.2 on page 116for more detailed discussion of the routine’s behavior.

GRsetchunkcache returns the value of the parameter maxcache if successful and FAIL (or -1) otherwise. The GRsetchunkcache parameters are discussed further in Table 8L.

* GRsetchunk, GRgetchunkinfo, GRsetchunkcache, GRwritechunk, and   
   GRreadchunk Parameter Lists

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Routine Name  [Return Type]  (FORTRAN-77) | Parameter | Parameter Type | | Description |
| C | FORTRAN-77 |
| GRsetchunk  [intn]  (mgschnk) | ri\_id | int32 | integer | Raster image identifier |
| c\_def | HDF\_CHUNK\_DEF | N/A | Chunk definition |
| flags | int32\* | N/A | Compression flags |
| dim\_length | N/A | integer | Chunk dimensions array |
| comp\_type | N/A | integer | Type of compression |
| comp\_prm | N/A | integer | Compression parameters array |
| GRgetchunkinfo  [intn]  (mggichnk) | ri\_id | int32 | integer | Raster image identifier |
| c\_def | HDF\_CHUNK\_DEF | N/A | Chunk definition |
| dim\_length | N/A | integer | Chunk dimensions array |
| flag | int32 | integer | Compression flag |
| GRsetchunkcache  [intn]  (mgscchnk) | ri\_id | int32 | integer | Raster image identifier |
| maxcache | int32 | integer | Maximum number of chunks to cache |
| flags | int32 | integer | Flags determining routine behavior |
| GRreadchunk  (mgrchnk/  mgrcchnk) | ri\_id | int32 | integer | Raster image identifier |
| origin | int32 | integer | Array specifying the coordinates of the chunk |
| datap | VOIDP | <valid\_numeric\_or\_  char\_data\_type> | Buffer with chunk data in pixel interlace mode |
| GRwritechunk  [intn]  (mgwchnk/  mgwcchnk) | ri\_id | int32 | integer | Raster image identifier |
| origin | int32 | integer | Array specifying the coordinates of the chunk |
| datap | const VOIDP | <valid\_numeric\_or\_  char\_data\_type> | Buffer with chunk data in pixel interlace mode |