## GRattrinfo/mgatinf

intn GRattrinfo(int32 *[obj]_id*, int32 *attr_index*, char **name*, int32 **data_type*, int32 **count*)

| | | |
|---|---|---|
| *[obj]_id* | IN: | Raster image identifier (*ri_id*), returned by **GRcreate** or **GRselect**, or GR interface identifier (*gr_id*), returned by **GRstart** |
| *attr_index* | IN: | Index of the attribute |
| *name* | OUT: | Buffer for the name of the attribute |
| *data_type* | OUT: | Data type of the attribute |
| *count* | OUT: | Number of attribute values |

**Purpose** Retrieves information about an attribute.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRattrinfo** retrieves the name, data type, and number of values of the attribute, specified by its index, *attr_index*, for the data object identified by the parameter *obj_id*. The name is stored in the parameter *name*, the data type is stored in the parameter *data_type*, and the number of values is stored in the parameter *count*. If the value of any of the output parameters is `NULL`, the corresponding information will not be retrieved.

The value of the parameter *attr_index* can be obtained using **GRfindattr**, **GRnametoindex** or **GRreftoindex**, depending on available information. Valid values of *attr_index* range from 0 to the total number of attributes attached to the object - 1. The total number of attributes attached to the file can be obtained using the routine **GRfileinfo**. The total number of attributes attached to an image can be obtained using the routine **GRgetiminfo** .

FORTRAN

```
integer function mgatinf([obj]_id, attr_index, name, data_type,
                            count)


integer [obj]_id, data_type, attr_index, count

character*(*) name
```

## GRcreate/mgcreat

int32 GRcreate(int32 *gr_id*, char **name*, int32 *ncomp*, int32 *data_type*, int32 *interlace_mode*, int32 *dim_sizes*[2])

| | | |
|---|---|---|
| *gr_id* | IN: | GR interface identifier returned by **GRstart** |
| *name* | IN: | Name of the raster image |
| *ncomp* | IN: | Number of pixel components in the image |
| *data_type* | IN: | Type of the image data |
| *interlace_mode* | IN: | Interlace mode of the image data |
| *dim_sizes* | IN: | Size of each dimension of the image |

**Purpose** Creates a new raster image.

**Return value** Returns a raster image identifier if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRcreate** creates a raster image with the values provided in the parameters *name*, *ncomp*, *data_type*, *interlace_mode* and *dim_sizes*.

The parameter *name* specifies the name of the image and must not be `NULL`. The length of the name should not be longer than `MAX_GR_NAME` (or `256`).

The parameter *ncomp* specifies the number of pixel components in the raster image and must have a value of at least 1.

The parameter *data_type* specifies the type of the raster image data and can be any of the data types supported by the HDF library. The data types supported by HDF are listed in Table 1A in Section I of this manual.

The parameter *interlace_mode* specifies the interlacing in which the raster image 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`).

The array *dimsizes* specifies the size of the two dimensions of the image. The dimensions must be specified and their values must be greater than `0`.

Once a raster image has been created, it is not possible to change its name, data type, dimension sizes or number of pixel components. However, it is possible to create a raster image and close the file before writing any data values to it. Later, the values can be added to or modified in the raster image, which then can be obtained using **GRselect**.

FORTRAN

```
integer function mgcreat(gr_id, name, ncomp, data_type,
                                   interlace_mode, dim_sizes)
```

```
integer gr_id, data_type, interlace_mode, ncomp, dim_sizes(2)

character*(*) name
```

## GRend/mgend

intn GRend(int32 *gr_id*)

| | | |
|---|---|---|
| *gr_id* | IN: | GR interface identifier returned by **GRstart** |

**Purpose** Terminates the GR interface session.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRend** terminates the GR interface session identified by the parameter *gr_id*.

**GRend**, together with **GRstart**, defines the extent of a GR interface session. **GRend** disposes of the internal structures initialized by the corresponding call to **GRstart**. There must be a call to **GRend** for each call to **GRstart**; failing to provide one may cause loss of data.

**GRstart** and **GRend** do not manage file access; use **Hopen** and **Hclose** to open and close HDF files. **Hopen** must be called before **GRstart** and **Hclose** must be called after **GRend**.

FORTRAN

```
integer function mgend(gr_id)


integer gr_id
```

## GRendaccess/mgendac

intn GRendaccess(int32 *ri_id*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |

**Purpose** Terminates access to a raster image.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRendaccess** terminates access to the raster image identified by the parameter *ri_id* and disposes of the raster image identifier. This access is initiated by either **GRselect** or **GRcreate**. There must be a call to **GRendaccess** for each call to **GRselect** or **GRcreate**; failing to provide this will result in loss of data. Attempts to access a raster image identifier disposed of by **GRendaccess** will result in an error condition.

FORTRAN

```
integer function mgendac(ri_id)


integer ri_id
```

## GRfileinfo/mgfinfo

intn GRfileinfo(int32 *gr_id*, int32 **n_images*, int32 **n_file_attrs*)

| | | |
|---|---|---|
| *gr_id* | IN: | GR interface identifier returned by **GRstart** |
| *n_images* | OUT: | Number of raster images in the file |
| *n_file_attrs* | OUT: | Number of global attributes in the file |

**Purpose** Retrieves the number of raster images and the number of global attributes in the file.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRfileinfo** retrieves the number of raster images and the number of global attributes for the GR interface identified by the parameter *gr_id*, and stores them into the parameters *n_images* and *n_file_attrs*, respectively.

The term “global attributes” refers to attributes that are assigned to the file instead of individual raster images. These attributes are created by **GRsetattr** with the object identifier parameter set to a GR interface identifier (*gr_id*) rather than a raster image identifier (*ri_id*).

**GRfileinfo** is useful in finding the range of acceptable indices for **GRselect** calls.

FORTRAN

```
integer function mgfinfo(gr_id, n_images, n_file_attrs)


integer gr_id, n_images, n_file_attrs
```

## GRfindattr/mgfndat

int32 GRfindattr(int32 *[obj]_id*, char **attr_name*)

| | | |
|---|---|---|
| *[obj]_id* | IN: | Raster image identifier (*ri_id*), returned by **GRcreate** or **GRselect**, or GR interface identifier (*gr_id*), returned by **GRstart** |
| *attr_name* | IN: | Name of the attribute |

**Purpose** Finds the index of a data object's attribute given an attribute name.

**Return value** Returns the index of the attribute if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRfindattr** returns the index of the attribute whose name is specified by the parameter *attr_name* for the object identified by the parameter *obj_id*.

FORTRAN

```
integer function mgfndat([obj]_id, attr_name)

integer [obj]_id

character*(*) attr_name
```

## GRgetattr/mggnatt/mggcatt

intn GRgetattr(int32 *[obj]_id*, int32 *attr_index*, VOIDP *values*)

| | | |
|---|---|---|
| *[obj]_id* | IN: | Raster image identifier (*ri_id*), returned by **GRcreate** or **GRselect**, or GR interface identifier (*gr_id*), returned by **GRstart** |
| *attr_index* | IN: | Index of the attribute |
| *values* | OUT: | Buffer for the attribute values |

**Purpose** Reads the values of an attribute for a data object.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRgetattr** obtains all values of the attribute that is specified by its index, *attr_index*, and is attached to the object identified by the parameter *obj_id*. The values are stored in the buffer *values*.

The value of the parameter *attr_index* can be obtained by using **GRfindattr**, **GRnametoindex**, or **GRreftoindex**, depending on available information. Valid values of *attr_index* range from 0 to the total number of attributes of the object - 1. The total number of attributes attached to the file can be obtained using the routine **GRfileinfo**. The total number of attributes attached to the image can be obtained using the routine **GRgetiminfo**.

**GRgetattr** only reads all values assigned to the attribute and not a subset.

Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**mggnatt**) and the other for character data (**mggcatt**).

FORTRAN

```
integer function mggnatt([obj]_id, attr_index, values)


integer [obj]_id, attr_index

<valid numeric data type> values(*)


integer function mggcatt([obj]_id, attr_index, values)


integer [obj]_id, attr_index

character*(*) values
```

## GRgetchunkinfo/mggichnk

intn GRgetchunkinfo(int32 *ri_id*, HDF_CHUNK_DEF **cdef*, int32 **flag*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |

C only:

| | | |
|---|---|---|
| *cdef* | OUT: | Pointer to the chunk definition |
| *flag* | OUT: | Pointer to the compression flag |

Fortran only:

| | | |
|---|---|---|
| *dim_length* | OUT: | Array of chunk dimensions |
| *flag* | OUT: | Compression flag |

**Purpose** Retrieves chunking information for a raster image.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRgetchunkinfo** retrieves chunking information about the raster image identified by the parameter *ri_id* into the parameters *cdef* and *flags* in C, and into the parameters *dim_length* and *flag* in Fortran. Note that only chunk dimensions are retrieved, compression information is not available.

The value returned in the parameter *flag* indicates if the raster image is not chunked, chunked, or chunked and compressed. The following table shows the possible values of the parameter *flag* and the corresponding characteristics of the raster image.

| Values of *flag* in C | Values of *flag* in Fortran | Raster Image Characteristics |
|---|---|---|
| HDF_NONE | -1 | Not chunked |
| HDF_CHUNK | 0 | Chunked and not compressed |
| HDF_CHUNK \| HDF_COMP | 1 | Chunked and compressed with either the run-length encoding (RLE), Skipping Huffman or GZIP compression algorithms |

In C, if the raster image is chunked and not compressed, **GRgetchunkinfo** fills the array `chunk_lengths` in the union `cdef` 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 `cdef` with the values of the corresponding chunk dimensions. Refer to the page on **GRsetchunk** in this manual for specific information on the union `HDF_CHUNK_DEF`. In Fortran, chunk dimensions are retrieved into the array `dim_length`. If the chunk length for each dimension is not needed, `NULL` can be passed in as the value of the parameter `cdef` in C.

FORTRAN

```
integer function mggichnk(ri_id, dim_length, flag)


integer ri_id, dim_length, flag
```

## GRgetiminfo/mggiinf

intn GRgetiminfo(int32 *ri_id*, char **gr_name*, int32 **ncomp*, int32 **data_type*, int32 **interlace_mode*, int32 *dim_sizes*[2], int32 **num_attrs*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *gr_name* | OUT: | Buffer for the name of the raster image |
| *ncomp* | OUT: | Number of components in the raster image |
| *data_type* | OUT: | Data type of the raster image data |
| *interlace_mode* | OUT: | Interlace mode of the stored raster image data |
| *dim_sizes* | OUT: | Sizes of raster image dimension |
| *num_attrs* | OUT: | Number of attributes attached to the raster image |

**Purpose** Retrieves general information about a raster image.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRgetiminfo** retrieves the name, number of components, data type, interlace mode, dimension sizes, and number of attributes of the raster image identified by the parameter *ri_id*.

**GRgetiminfo** stores the name, number of components, data type, interlace mode and dimension sizes of the image in the parameters *gr_name*, *ncomp*, *data_type*, *interlace_mode*, and *dim_sizes*, respectively. It also retrieves the number of attributes attached to the image into the parameter *num_attrs*. If the value of any of the output parameters are set to `NULL` in C, the corresponding information will not be retrieved.

The buffer *gr_name* is assumed to have sufficient space allocated to store the entire name of the raster image.

The valid values of the parameter *data_type* are listed in Table 1A in Section I of this manual.

FORTRAN

```
integer function mggiinf(ri_id, gr_name, ncomp, data_type,
                                interlace_mode, dim_sizes, num_attrs)


integer ri_id, ncomp, data_type, interlace_mode, num_attrs

integer dim_sizes[2]

character*(*) gr_name
```

## GRgetlutid/mggltid

int32 GRgetlutid(int32 *ri_id*, int32 *pal_index*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *pal_index* | IN: | Index of the palette |

**Purpose** Gets the identifier of a palette given its index.

**Return value** Returns the palette identifier if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRgetlutid** gets the identifier of the palette attached to the raster image identified by the parameter *ri_id*. The palette is identified by its index, *pal_index*.

Currently, only one palette can be assigned to a raster image, which means that *pal_index* should always be set to `0`.

FORTRAN

```
integer function mggltid(ri_id, pal_index)


integer ri_id, pal_index
```

## GRgetlutinfo/mgglinf

intn GRgetlutinfo(int32 *pal_id*, int32 **ncomp*, int32 **data_type*, int32 **interlace_mode*, int32 **num_entries*)

| | | |
|---|---|---|
| *pal_id* | IN: | Palette identifier returned by **GRgetlutid** |
| *ncomp* | OUT: | Number of components in the palette |
| *data_type* | OUT: | Data type of the palette |
| *interlace_mode* | OUT: | Interlace mode of the stored palette data |
| *num_entries* | OUT: | Number of color lookup table entries in the palette |

**Purpose** Retrieves information about a palette.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRgetlutinfo** retrieves the number of pixel components, data type, interlace mode, and number of color lookup table entries of the palette identified by the parameter *pal_id*. These values are stored in the parameters *ncomp*, *data_type*, *interlace_mode*, and *num_entries*, respectively. In C if the value of any of the output parameters are set to `NULL`, the corresponding information will not be retrieved.

FORTRAN

```
integer function mgglinf(pal_id, ncomp, data_type, interlace_mode,
                                     num_entries)

integer pal_id, ncomp, data_type, interlace_mode, num_entries
```

## GRidtoref/mgid2rf

uint16 GRidtoref(int32 *ri_id*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRselect** or **GRcreate** |

**Purpose** Maps a raster image identifier to a reference number.

**Return value** Returns the reference number of the raster image if successful and `0` otherwise.

**Description** **GRidtoref** returns the reference number of the raster image identified by the parameter *ri_id*.

This routine is commonly used for the purpose of annotating the raster image or including the raster image within a vgroup.

FORTRAN

```
integer function mgid2rf(ri_id)


integer ri_id
```

## GRluttoref/mglt2rf

uint16 GRluttoref(int32 *pal_id*)

| | | |
|---|---|---|
| *pal_id* | IN: | Palette identifier returned by **GRgetlutid** |

**Purpose** Maps a palette identifier to a reference number.

**Return value** Returns the reference number of the palette if successful or `0` otherwise.

**Description** **GRluttoref** returns the reference number of the palette identified by the parameter *pal_id*.

This routine is commonly used for the purpose of annotating the palette or including the palette within a vgroup.

FORTRAN

```
integer function mglt2rf(pal_id)


integer pal_id
```

## GRnametoindex/mgn2ndx

int32 GRnametoindex(int32 *gr_id*, char *gr_*name*)

| | | |
|---|---|---|
| *gr_id* | IN: | GR interface identifier returned by **GRstart** |
| *ri_name* | IN: | Name of the raster image |

**Purpose** Maps the name of a raster image to an index.

**Return value** Returns the index of the raster image if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRnametoindex** returns, for the GR interface identified by the parameter *gr_id*, the index (*index*) of the raster image named *gr_name*.

The value of *index* can be passed into **GRselect** to obtain the raster image identifier (*ri_id*).

FORTRAN

```
integer function mgn2ndx(gr_id, gr_name)


integer gr_id

character*(*) gr_name
```

## GRreadimage/mgrdimg/mgrcimg

intn GRreadimage(int32 *ri_id*, int32 *start*[2], int32 *stride*[2], int32 *edge*[2], VOIDP *data*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *start* | IN: | Array specifying the starting location from where raster image data is read |
| *stride* | IN: | Array specifying the interval between the values that will be read along each dimension |
| *edge* | IN: | Array specifying the number of values to be read along each dimension |
| *data* | OUT: | Buffer for the image data |

**Purpose** Reads a raster image.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRreadimage** reads the subsample of the raster image specified by the parameter *ri_id* into the buffer *data*. The subsample is defined by the values of the parameters *start*, *stride* and *edge*.

The array *start* specifies the starting location of the subsample to be read. 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. 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. For example, if the first value of the array *start* is 2 and the second value is 3, the starting location of the subsample to be read is at the fourth row and third column in C, and at the third row and fourth column in Fortran. Note that the correspondence between the elements in the array *start* and the array *data* dimensions in the GR interface is different from that in the SD interface. See the reference manual page on **SDreaddata** for an example.

The array *stride* specifies the reading 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 read. If one of the elements of the array *stride* is 2, then every other element along the corresponding dimension of the array *data* will be read, 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*.

Each element of the array *edges* specifies the number of data elements to be read along the corresponding dimension. The correspondence between the elements of the array *edges* and the dimensions of the array *data* is the same as described above for the array *start*.

Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**mgrdimg**) and the other for character data (**mgrcimg**).

FORTRAN

```
integer function mgrdimg(ri_id, start, stride, edge, data)

integer ri_id, start(2), stride(2), edge(2)

<valid numeric data type> data(*)
```

```
integer function mgrcimg(ri_id, start, stride, edge, data)

integer ri_id, start(2), stride(2), edge(2)

character*(*) data
```

## GRreadlut/mgrdlut/mgrclut

intn GRreadlut(int32 *pal_id*, VOIDP *pal_data*)

| | | |
|---|---|---|
| *pal_id* | IN: | Palette identifier returned by **GRgetlutid** |
| *pal_data* | OUT: | Buffer for the palette data |

**Purpose** Reads a palette.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRreadlut** reads the palette specified by the parameter *pal_id* into the buffer *pal_data*.

Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**mgrdlut**) and the other for character data (**mgrclut**).

FORTRAN

```
integer function mgrdlut(pal_id, pal_data)

integer pal_id

<valid numeric data type> pal_data(*)

integer function mgrclut(pal_id, pal_data)

integer pal_id

character*(*) pal_data
```

## GRreftoindex/mgr2idx

int32 GRreftoindex(int32 *gr_id*, uint16 *gr_ref*)

| | | |
|---|---|---|
| *gr_id* | IN: | GR interface identifier returned by **GRstart** |
| *gr_ref* | IN: | Reference number of the raster image |

**Purpose** Maps the reference number of a raster image to an index.

**Return value** Returns the index of the image if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRreftoindex** returns the index of the raster image specified by the parameter *gr_ref*.

FORTRAN

```
integer function mgr2idx(gr_id, gr_ref)


integer gr_id, gr_ref
```

## GRreqimageil/mgrimil

intn GRreqimageil(int32 *ri_id*, intn *interlace_mode*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *interlace_mode* | IN: | Interlace mode |

**Purpose** Specifies the interlace mode to be used in the subsequent raster image read operation(s).

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRreqimageil** requests that the subsequent read operations on the image identified by the parameter *ri_id* use the interlace mode specified by the parameter *interlace_mode.*

The parameter *interlace_mode* specifies the interlace mode in which the data will be stored in memory when being read. Valid values of the parameter *interlace_mode* are `MFGR_INTERLACE_PIXEL` (or `0`), `MFGR_INTERLACE_LINE` (or `1`) and `MFGR_INTERLACE_COMPONENT` (or `2`).

In the file, the image is always stored in pixel interlace mode, i.e. `MFGR_INTERLACE_PIXEL`. The interlace mode of the raster image specified at creation time is stored in the file along with the raster image. If **GRreqimageil** is not called prior to the call to **GRreadimage**, the raster image will be read and stored in memory according to the interlace mode specified at creation. If **GRreqimageil** is called before **GRreadimage**, **GRreadimage** will read the raster image and store it according to the interlace mode specified in the call to **GRreqimageil**.

FORTRAN

```
integer function mgrimil(ri_id, interlace_mode)


integer ri_id, interlace_mode
```

## GRreqlutil/mgrltil

intn GRreqlutil(int32 *ri_id*, intn *interlace_mode*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *interlace_mode* | IN: | Interlace mode |

**Purpose** Specifies the interlace mode to be used in the next palette read operation(s).

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRreqlutil** requests that the subsequent read operations on the palette attached to the image identified by the parameter *ri_id*, use the interlace mode *interlace_mode*.

The parameter *interlace_mode* specifies the interlace mode in which the data will be stored in memory when being read. Valid values of the parameter *interlace_mode* are `MFGR_INTERLACE_PIXEL` (or `0`), `MFGR_INTERLACE_LINE` (or `1`) and `MFGR_INTERLACE_COMPONENT` (or `2`).

FORTRAN

```
integer function mgrltil(ri_id, interlace_mode)


integer ri_id, interlace_mode
```

## GRselect/mgselct

int32 GRselect(int32 *gr_id*, int32 *index*)

| | | |
|---|---|---|
| *gr_id* | IN: | GR interface identifier returned by **GRstart** |
| *index* | IN: | Index of the raster image in the file |

**Purpose** Selects the existing raster image.

**Return value** Returns the raster image identifier if successful or `FAIL` (or `-1`) otherwise.

**Description** **GRselect** obtains the identifier of the raster image specified by the its index, *index*.

Valid values of the parameter *index* range from 0 to the total number of raster images in the file - 1. The total number of the raster images in the file can be obtained by using **GRfileinfo**.

FORTRAN

```
integer function mgselct(gr_id, index)


integer gr_id, index
```

## GRsetattr/mgsnatt/mgscatt

intn GRsetattr(int32 *[obj]_id*, char **attr_name*, int32 *data_type*, int32 *count*, VOIDP *values*)

| | | |
|---|---|---|
| *[obj]_id* | IN: | Raster image identifier (*ri_id*), returned by **GRcreate** or **GRselect** or GR interface identifier (*gr_id*), returned by **GRstart** |
| *attr_name* | IN: | Name of the attribute |
| *data_type* | IN: | Data type of the attribute |
| *count* | IN: | Number of values in the attribute |
| *values* | IN: | Buffer for the attribute values |

**Purpose** Assigns an attribute to a raster image or a file.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRsetattr** attaches the attribute to the object specified by the parameter *obj_id*. The attribute is defined by its name, *attr_name*, data type, *data_type*, number of attribute values, *count*, and the attribute values, *values*. **GRsetattr** provides a generic way for users to define metadata. It implements the label = value data abstraction.

If an GR interface identifier (*gr_id)* is specified as the parameter *obj_id*, a global attribute is created which applies to all objects in the file. If a raster image identifier (*ri_id*) is specified as the parameter *obj_id*, an attribute is attached to the specified raster image.

The parameter *attr_name* can be any ASCII string.

The parameter *data_type* can contain any data type supported by the HDF library. These data types are listed in Table 1A in Section I of this manual.

Attribute values are passed in the parameter *values*. The number of attribute values is defined by the parameter *count* . If more than one value is stored, all values must have the same data type. If an attribute with the given name, data type and number of values exists, it will be overwritten. Currently, the only predefined attribute is the fill value, identified by the `FILL_ATTR` definition.

Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**mgsnatt**) and the other for character data (**mgscatt**).

FORTRAN

```
integer function mgsnatt([obj]_id, attr_name, data_type, count,
                               values)


integer [obj]_id, data_type, count

character*(*) attr_name

<valid numeric data type> values(*)
```

```
integer function mgscatt([obj]_id, attr_name, data_type, count,
                            values)


integer [obj]_id, data_type

integer count

character*(*) values, attr_name
```

## GRsetcompress/mgscompress

*(**Note:** The GRsetcompress routine does not work in the current release.)*

intn GRsetcompress(int32 *ri_id*, int32 *comp_type*, comp_info **c_info*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *comp_type* | IN: | Compression method for the image data |

C only:

| | | |
|---|---|---|
| *c_info* | IN: | Pointer to the `comp_info` union |

Fortran only:

| | | |
|---|---|---|
| *comp_prm* | IN: | Compression parameters array |

**Purpose** Specifies if the raster image will be stored in a file as a compressed raster image.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRsetcompress** specifies if the raster image specified by the parameter *ri_id* will be stored in the file in compressed format.

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

The compression method parameters are specified by the parameter *c_info* in C and the parameter *comp_prm* in Fortran. The parameter *c_info* has type `comp_info`, which is described in the hcomp.h header file. It contains algorithm-specific information for the library compression routines.

The skipping size for the Skipping Huffman algorithm is specified in the field `c_info.skphuff.skp_size` in C and in the parameter *comp_prm*(1) in Fortran.

The deflate level for the GZIP algorithm is specified in the field `c_info.deflate.level` in C and in the parameter *comp_prm*(1) in the Fortran.

FORTRAN

```
integer mgscompress(ri_id, comp_type, comp_prm)

integer ri_id, comp_type, comp_prm(*)
```

## GRsetchunk/mgschnk

intn GRsetchunk(int32 *ri_id*, HDF_CHUNK_DEF *cdef*, int32 *flags*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| C only: | | |
| *cdef* | IN: | Chunk definition |
| *flags* | IN: | Compression flags |
| Fortran only: | | |
| *dim_length* | IN: | Chunk dimensions array |
| *comp_type* | IN: | Type of compression |
| *comp_prm* | IN: | Compression parameters array |

**Purpose** Makes a raster image a chunked raster image.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRsetchunk** makes the raster image specified by the parameter *ri_id* a chunked raster image according to the chunking and compression information provided in the parameters *cdef* and *flags* in C, or in the parameters *comp_type* and *comp_prm* in Fortran.

### C only:

The parameter *cdef* 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 or GZIP compression algorithms.

If the parameter *flags* has a value of `HDF_CHUNK`, the chunk dimensions must be specified in the field `cdef.chunk_lengths[]`. If the parameter *flags* has a value of (`HDF_CHUNK` | `HDF_COMP`), the following must be specified:

1) The chunk dimensions in the field `cdef.comp.chunk_lengths[].`
2) The compression type in the field `cdef.comp.comp_type`. Valid values of compression type values are listed below.

`COMP_CODE_NONE` (or `0`) for uncompressed data
`COMP_CODE_RLE` (or `1`) for data compressed using the RLE compression algorithm
`COMP_CODE_SKPHUFF` (or `3`) for data compressed using the Skipping Huffman compression algorithm
`COMP_CODE_DEFLATE` (or `4`) for data compressed using the GZIP compression algorithm

3) If using Skipping Huffman compression, the skipping size is specified in the field `cdef.comp.cinfo.skphuff.skp_size`. If using GZIP compression, the deflate level is specified in the field `cdef.comp.cinfo.deflate.level`. Valid deflate level values are integers from 1 to 9 inclusive.

Refer to the **SDsetcompress** page in this manual for the definition of the `comp_info` structure.

**Fortran only:**

The *dim_length* array specifies the chunk dimensions.

The parameter *comp_type* specifies the compression type. Valid compression types and their values used 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 data compressed using the RLE compression algorithm
`COMP_CODE_SKPHUFF` (or `3`) for data compressed using the Skipping Huffman compression algorithm
`COMP_CODE_DEFLATE` (or `4`) for data compressed using the GZIP compression algorithm.

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

FORTRAN
```
integer function mgschnk(ri_id, dim_length, comp_type, comp_prm)
integer ri_id, dim_length, comp_type, comp_prm
```

## GRsetchunkcache/mgscchnk

intn GRsetchunkcache(int32 *ri_id*, int32 *maxcache*, int32 *flags*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *maxcache* | IN: | Maximum number of chunks to cache |
| *flags* | IN: | Flags determining the behavior of the routine |

**Purpose** Specifies the maximum number of chunks to cache.

**Return value** Returns the value of the parameter *maxcache* if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRsetchunkcache** sets the maximum number of chunks to be cached for the chunked raster image specified by the parameter *ri_id*. The maximum number of the 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. Refer to the discussion of the **GRsetchunkcache** routine in the *HDF User's Guide* for more specific information on the routine's behavior.

FORTRAN

```
integer function mgscchnk(ri_id, maxcache, flags)


integer ri_id, maxcache, flags
```

## GRsetexternalfile/mgsxfil

intn GRsetexternalfile(int32 *ri_id*, char **filename*, int32 *offset*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *filename* | IN: | Name of the external file |
| *offset* | IN: | Offset in bytes from the beginning of the external file to where the data will be written |

**Purpose** Specifies that the raster image will be written to an external file.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRsetexternalfile** specifies that the raster image identified by the parameter *ri_id* will be written to the external file specified by the parameter *filename* at the offset specified by the parameter *offset*.

Data can only be moved once for any given raster image, and it is the user's responsibility to make sure the external data file is kept with the “original” file.

If the raster image already exists, its data will be moved to the external file . Space occupied by the data in the primary file will not be released. To release the space in the primary file use the `hdfpack` command-line utility. If the raster image does not exist, its data will be written to the external file during the subsequent calls to **GRwritedata**.

See the reference manual entries for **HXsetcreatedir** and **HXsetdir** for more information on the options available for accessing external files.

FORTRAN

```
integer function mgsxfil(ri_id, filename, offset)


integer ri_id, offset

character*(*) filename
```

## GRstart/mgstart

int32 GRstart(int32 *file_id*)

| | | |
|---|---|---|
| *file_id* | IN: | File identifier returned by **Hopen** |

**Purpose** Initializes the GR interface.

**Return value** Returns the GR interface identifier if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRstart** initializes the GR interface for the file specified by the parameter *file_id*.

This routine is used with the **GRend** routine to define the extent of the GR interface session. As with the start routines in the other interfaces, **GRstart** initializes the internal interface structures needed for the remaining GR routines. Use the general purpose routines **Hopen** and **Hclose** to manage file access. The GR routines will not open and close HDF files.

FORTRAN

```
integer function mgstart(file_id)


integer file_id
```

## GRwriteimage/mgwrimg/mgwcimg

intn GRwriteimage(int32 *ri_id*, int32 *start*[2], int32 *stride*[2], int32 *edge*[2], VOIDP *data*)

| | | |
|---|---|---|
| *ri_id* | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| *start* | IN: | Array containing the two-dimensional coordinate of the initial location for the write |
| *stride* | IN: | Array containing the number of data locations the current location is to be moved forward before each write |
| *edge* | IN: | Array containing the number of data elements that will be written along each dimension |
| *data* | IN: | Buffer containing the image data |

**Purpose** Writes a raster image.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRwriteimage** writes the subsample of the raster image data stored in the buffer *data* to the raster image specified by the parameter *ri_id*. The subsample is defined by the values of the parameters *start*, *stride* and *edge.*

The array *start* specifies the starting location of the subsample 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. 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. For example, if the first value of the array *start* is 2 and the second value is 3, the starting location of the subsample to be written is at the fourth row and third column in C, and at the third row and fourth column in Fortran. 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 the reference manual page on **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*.

Each element of the array *edges* specifies the number of data elements to be written along the corresponding dimension. The correspondence between the elements of the array *edges* and the dimensions of the array *data* is the same as described above for the array *start*.

Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**mgwrimg**) and the other for character data (**mgwcimg**).

FORTRAN

```
integer function mgwrimg(ri_id, start, stride, edge, data)


integer ri_id, start(2), stride(2), edge(2)

<valid numeric data type> data(*)


integer function mgwcimg(ri_id, start, stride, edge, data)


integer ri_id, start(2), stride(2), edge(2)

character*(*) data
```

## GRwritelut/mgwrlut/mgwclut

intn GRwritetlut(int32 *pal_id*, int32 *ncomp*, int32 *data_type*, int32 *interlace_mode*, int32 *num_entries*,
VOIDP *pal_data*)

| | | |
|---|---|---|
| *pal_id* | IN: | Palette identifier returned by **GRgetlutid** |
| *ncomp* | IN: | Number of components in the palette |
| *data_type* | IN: | Data type of the palette data |
| *interlace_mode* | IN: | Interlace mode of the stored palette data |
| *num_entries* | IN: | Number of entries in the palette |
| *pal_data* | IN: | Buffer for the palette data to be written |

**Purpose** Writes a palette.

**Return value** Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

**Description** **GRwritelut** writes a palette with the number of pixel components specified by the parameter *ncomp*, the data type of the palette data specified by the parameter *data_type*, the interlace mode specified by the parameter *interlace_mode*, and the number of entries in the palette specified by the parameter *num_entries*. The palette data itself is stored in the *pal_data* buffer. **Currently only "old-style" palettes are supported**, i.e *ncomp = 3, num_entries = 256, data_type = uint8.*

The parameter *ncomp* specifies the number of pixel components in the palette and must have a value of at least 1.

The parameter *data_type* specifies the type of the palette data and can be any of the data types supported by the HDF library. The data types supported by HDF are listed in Table 1A in Section I of this manual.

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`).

The buffer *pal_data* is assumed to have sufficient space allocated to store all of the palette data.

Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**mgwrlut**) and the other for character data (**mgwclut**).

FORTRAN

```
integer function mgwrlut(pal_id, ncomp, data_type, interlace_mode,
                                num_entries, pal_data)


integer pal_id, ncomp, data_type, interlace_mode, num_entries

<valid numeric data type> pal_data(*)
```

```
integer function mgwclut(pal_id, ncomp, data_type, interlace_mode,
                          num_entries, pal_data)

integer pal_id, ncomp, data_type, interlace_mode, num_entries

character*(*) pal_data
```