GRattrinfo/mgatinf

intn GRattrinfo(int32 *[obj]\_id*, int32 *attr\_index*, char \**name*, int32 \**attr\_nt*, 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 |
| attr\_nt | OUT: | Number 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 *attr\_nt*, 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, attr\_nt, count) | | |
|  | integer [obj]\_id, attr\_nt, attr\_index, count | | |
|  | character\*(\*) name | | |

GRcreate/mgcreat

int32 GRcreate(int32 *gr\_id*, char *\*name*,int32 *ncomp*,int32 *nt*,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 |
| nt | IN: | Number 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, nt, 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 nt specifies the type of the raster image data and can be any of the number types supported by the HDF library and 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, 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. | | |
|  | 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. | | |
| Note | **Regarding an important difference between the SD and GR interfaces:** 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 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.  **o** 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.  **o** 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. | | |
| 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 | | |

GRgetattdatainfo

intn GRgetattdatainfo(int32 *obj\_id*, int32 *attr\_index,* int32 *\*offset,* int32 *\*length*)

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Raster image identifier (*ri\_id*), returned by GRselect, or GR interface identifier (*gr\_id*), returned by GRstart |
| attr\_index | IN: | Index of the inquired attribute |
| offset | OUT: | Buffer to hold offset of the attribute’s data |
| length | OUT: | Buffer to hold length of the attribute’s data |
| Purpose | Retrieves location and size of attribute's data. | | |
| Return value | Returns the number of data blocks retrieved, which should be 1, if successful, and FAIL (or -1) otherwise. | | |
| Description | GRgetattdatainfo retrieves the offset and length of the data that belongs to the attribute attr\_index, which is attached to the HDF4 object specified by obj\_id. The value of obj\_id can be a GR interface identifier (gr\_id), returned by GRstart or an image identifier (ri\_id), returned by GRselect. | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |

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 cdefwith 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 cdefwith 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 | | | | | |

GRgetcompinfo/mggcompress

intn GRgetcompinfo(int32 *ri\_id*, comp\_coder\_t \**comp\_type*, comp\_info \**c\_info*)

|  |  |  |
| --- | --- | --- |
| ri\_id | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| comp\_type | OUT: | Type of compression |
| *C only:* c\_info | OUT: | Pointer to compression information structure |
| *Fortran only:* comp\_prm | OUT: | Compression parameters array |
| Purpose | Retrieves raster image data compression type and compression information. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | GRgetcompinfo retrieves the compression type and compression information for the specified raster image. GRgetcompinfo replaces GRgetcompress because this function has flaws, causing failure for some chunked and chunked/compressed data. | | |
|  | The compression method is returned in the parameter comp\_type. Valid values of comp\_type 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  COMP\_CODE\_JPEG (or 7) for JPEG compression  COMP\_CODE\_IMCOMP (or 12) for IMCOMP compression | | |
|  | When a compression method requires additional parameters, those values are returned in the c\_info struct in C and the array parameter comp\_prm in Fortran. | | |
|  | 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 and in the GRsetcompress entry in this reference manual. | | |
|  | The *comp\_prm* parameter is an array of one element:  **o** With Skipping Huffman compression, comp\_prm(1) contains the skip value, skphuff\_skp\_size.  **o** In the case of GZIP compression, comp\_prm(1) contains the deflation value, deflate\_value.  **o** comp\_prmis ignored with other compression methods. (There are no relevant RLE parameters and the quality and force\_baseline data are not available for JPEG images. If GRgetcompinfo is called for either an RLE or a JPEG image, the function will return only the compression type; c\_info will contain only zeros.)  **o** Currently, Fortran GR interface doesn’t support Szip compression. | | |
| FORTRAN | integer function mggcompress(ri\_id, comp\_type, comp\_prm) | | |
|  | integer ri\_id, comp\_type, comp\_prm(1) | | |

GRgetcomptype

intn GRgetcomptype(int32 *ri\_id*, comp\_coder\_t \**comp\_type*)

|  |  |  |
| --- | --- | --- |
| ri\_id | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| comp\_type | OUT: | Type of compression |
| Purpose | Retrieves the compression type of a raster image's data. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | GRgetcomptype retrieves the compression type for the specified raster image. | | |
|  | The compression type is returned in the parameter comp\_type. Valid values of comp\_type 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  COMP\_CODE\_JPEG (or 7) for JPEG compression  COMP\_CODE\_IMCOMP (or 12) for IMCOMP compression | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |

GRgetdatainfo

intn GRgetdatainfo(int32 *ri\_id*, uintn *start\_block*, uintn *info\_count*, int32 \**offsetarray*, int32 *\*lengtharray*)

|  |  |  |
| --- | --- | --- |
| ri\_id | IN: | Raster image identifier returned by GRselect |
| start\_block | IN: | Value indicating where to start reading offsets |
| info\_count | IN: | Length of the offset and length lists |
| offsetarray | OUT: | Array to hold offsets of the data blocks |
| lengtharray | OUT: | Array to hold lengths of the data blocks |
| Purpose | Retrieves location and size of data blocks in a specified raster image, starting at a specified block. | | |
| Return value | Returns the number of data blocks retrieved if successful, and FAIL (or -1) otherwise. | | |
| Description | GRgetdatainfo retrieves two lists, offsetarray and lengtharray, containing the offsets and lengths of the blocks of data belonging to the raster image specified by ri\_id. | | |
|  | The parameter info\_count provides the number of offset/length values that the lists can hold. To allocate sufficient memory for offsetarray and lengtharray, the application can invoke GRgetdatainfo passing in 0 for info\_count and NULL for both arrays to get the value for info\_count in the next call to GRgetdatainfo. | | |
|  | The parameter start\_block is an integer value between 0 and number of blocks - 1. The combination of parameters info\_length and start\_block provide user applications with flexibility of where and how much data information to retrieve. | | |
|  | o When start\_block is 0, GRgetdatainfo will start getting data info from the beginning of the image's data.  o When start\_block is greater than the number of blocks in the image, GRgetdatainfo will return FAIL (or -1). | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |

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 |
| nt | OUT: | Number 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, number 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, number type, interlace mode and dimension sizes of the image in the parameters gr\_name*,* ncomp*,* nt*,* 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 nt 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 |
| nt | OUT: | Number 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, number 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, nt, 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, nt, interlace\_mode, num\_entries) | | |
|  | integer pal\_id, ncomp, nt, interlace\_mode, num\_entries | | |

GRgetnluts/mggnluts

intn GRgetnluts(int32 *ri\_id*)

|  |  |  |
| --- | --- | --- |
| ri\_id | IN: | Data set identifier returned by **GRcreate** or **GRselect** |
| Purpose | Retrieves the number of palettes for an image. | | |
| Return value | Returns number of palettes (1 or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **GRgetnluts** retrieves the number of palettes (or color look-up tables, commonly abbreviated as LUTs) available for the specified raster image, ri\_id. | | |
|  | There can currently be either 0 or 1 palettes assigned to an image. If multiple palettes are supported in a future release, this function may then return values greater than 1. | | |
| FORTRAN | integer function mggnluts(ri\_id) | | |
|  | integer ri\_id | | |

GRgetpalinfo

intn GRgetpalinfo(int32 gr\_id, uintn pal\_count; hdf\_ddinfo\_t \*palinfo\_array)

|  |  |  |
| --- | --- | --- |
| gr\_id | IN: | GR identifier for the file, returned by **GRstart** |
| pal\_count | IN | Length of the palette data descriptor (DD) array |
| palinfo\_array | IN/OUT | Array containing palettes’ data descriptor (DD) information |

|  |  |
| --- | --- |
| Purpose | Retrieves data descriptor information for palettes in a file, i.e., tag, reference number, offset, and length. |
| Return value | Returns the number of palette data descriptors retrieved if successful and FAIL (or -1) otherwise. |
| Description | GRgetpalinfo retrieves a list of structures containing the data descriptors (DD) of the palettes in the file, specified by gr\_id. Each palette DD contains a palette tag, reference number, offset, and length together specifying the palette’s data. |
|  | The argument pal\_count specifies the size of the list. GRgetpalinfo returns the number of palette data descriptors (DDs) in the file when called with 0 for the pal\_count and NULL for the palinfo\_array. DDs with the tags DFTAG\_IP8 and DFTAG\_LUT are counted. If the function is not successful, FAIL will be returned. |
|  | When pal\_count is a positive number and palinfo\_array is not NULL, GRgetpalinfo will populate palinfo\_array with the palette data descriptor tag/ref pairs and the offsets and lengths of the corresponding palette data elements in the file. The palinfo\_array must be allocated sufficiently to hold all the descriptor information to be retrieved, as specified by pal\_count. The function will return the number of palette DDs retrieved if successful and FAIL, otherwise. |
|  | After palinfo\_array has been populated, an application can use the tag/ref values of each entry in the array as parameters to Hgetelement to retrieve the palette data associated with each palette DD in the HDF4 file. |
|  | The structure hdf\_ddinfo\_t is defined as:  typedef struct hdf\_ddinfo  {  uint16tag;/\* palette tag \*/  uint16ref; /\* palette ref \*/  int32offset; /\* position of the palette data \*/  int32length; /\* length of the palette data \*/  } hdf\_ddinfo\_t; |

|  |  |
| --- | --- |
| FORTRAN | Currently unavailable |
|  |  |

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 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 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 \**ri\_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 converts the name of a raster image, ri\_name, to an index (*index*) in the GR file, identified by gr\_id. | | |
|  | The value of *index* can be passed into GRselect to obtain the raster image identifier (*ri\_id*). | | |
| FORTRAN | integer function mgn2ndx(gr\_id, ri\_name) | | |
|  | integer gr\_id | | |
|  | character\*(\*) ri\_name | | |

GRreadchunk/mgrchnk/mgrcchnk

intn GRreadchunk(int32 ri\_id, int32 \*origin, VOIDP datap)

|  |  |  |
| --- | --- | --- |
| *ri\_id* | IN: | Raster image identifier returned by GRcreate or GRselect |
| *origin* | IN: | Origin of the chunk to be read |
| *datap* | IN: | Buffer for the chunk to be read |
| Purpose | Reads a data chunk from a chunked raster image (pixel-interlace only) | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | GRreadchunk reads the entire chunk of data from the chunked raster image identified by ri\_id and stores it in the buffer datap. 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. | | |
|  | GRreadchunk will return FAIL (or -1) when an attempt is made to use it to read from a non-chunked raster image. | | |
|  | 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. Refer to Chapter 8, "General Raster Images (GR API)" in the HDF User’s Guide for details. | | |
|  | The buffer datap contains the chunk data organized in pixel interlace mode. | | |
| FORTRAN | integer mgrchnk(ri\_id, origin, datap) | | |
|  | integer ri\_id, origin(2) | | |
|  | <valid\_numeric\_datatype> datap(\*) | | |
|  |  | | |
|  | integer mgrcchnk(ri\_id, origin, char\_datap) | | |
|  | integer ri\_id, origin(2) | | |
|  | character\*(\*) char\_datap | | |

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 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. | | |
|  | 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, if there were any un-written elements in the image, they would have been filled with the image’s fill value, which could be provided prior to writing the image, or the default fill-value, which is 0. | | |
|  | Note that there are two FORTRAN-77 versions of this routine; one for numeric data (mgrdimg) and the other for character data (mgrcimg). | | |
| Note | **Regarding an important difference between the SD and GR interfaces:** 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 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.  **o** 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.  **o** 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. | | |
|  |  | | |
| 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 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 *ri\_ref*)

|  |  |  |
| --- | --- | --- |
| gr\_id | IN: | GR interface identifier returned by GRstart |
| ri\_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 its reference number ri\_ref, in the GR file identified by gr\_id. | | |
| FORTRAN | integer function mgr2idx(gr\_id, ri\_ref) | | |
|  | integer gr\_id, ri\_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 | | |

GRsetaccesstype/mgsactp

intn GRsetaccesstype(int32 *ri\_id*, uintn *accesstype*)

|  |  |  |
| --- | --- | --- |
| ri\_id | IN: | Raster image identifier returned by **GRcreate** or **GRselect** |
| accesstype | IN: | Access type |
| Purpose | Sets the access for an RI to be either serial or parallel I/O. | | |
| Return value | Returns SUCCEED (or 0) if the RI data can be accessed via accesstype and FAIL (or -1) otherwise. | | |
| Description | **GRsetaccesstype** sets the access type to be either serial or parallel I/O for the raster image specified by ri\_id. Access types can be DFACC\_SERIAL (or 1), DFACC\_PARALLEL (or 11), or DFACC\_DEFAULT (or 0). | | |
| FORTRAN | integer function mgsactp(ri\_id, accesstype) | | |
|  | integer ri\_id, accesstype | | |

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 |
| attr\_nt | IN: | Number 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 an attribute to the object specified by obj\_id. The attribute is specified by its name, attr\_name, number type, attr\_nt, number of attribute values, count, and the attribute values, values. GRsetattr provides a generic way for users to define metadata in the GR interface. 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 with maximum length of H4\_MAX\_NC\_NAME (or 256). | | |
|  | The parameter attr\_nt 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 if an image does not have a fill value defined, and is written partially, a FILL\_ATTR attribute will be added. This attribute has a value of 0, which is the image’s fill value. Thus, any un-written elements in the image will be filled with the default fill value of 0. | | |
|  | 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 ri\_id, comp\_type, comp\_prm(\*) | | |
|  | 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 | | |

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; /\* compression information \*/  } comp;  struct  {  int32 chunk\_lengths[2];  intn start\_bit;  intn bit\_len;  intn sign\_ext;  intn fill\_one;  } 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 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 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 and GZIP compression, parameters are passed in corresponding fields of the structure cinfo.  o Specify skipping size for Skipping Huffman compression in the field cdef.comp.cinfo.skphuff.skp\_size, which must be an integer of value 1 or greater.  o Specify the deflate level for GZIP compression in the field cdef.comp.cinfo.deflate\_level. Valid deflate level values are integers between 0 and 9 inclusive.  o Specify the options mask and the number of pixels per block for Szip compression in the fields c\_info.szip.options\_mask and c\_info.szip.pixels\_per\_block, respectively.  Refer to the SDsetcompress entry in this reference manual for details on these parameters. | | |
|  | 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 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, Fortran GR interface does not support Szip 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 | | |

GRsetcompress/mgscompress

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 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\_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  COMP\_CODE\_JPEG (or 7) for JPEG 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 Fortran. | | |
|  | The parameter c\_info is a pointer to a union structure of type comp\_info. This union structure is defined as follows: | | |
|  | typedef union tag\_comp\_info  {  struct  { /\* Not used by GRsetcompress \*/ } jpeg;  struct  { /\* Not used by GRsetcompress \*/ } nbit;  struct  { /\* struct to contain info about how to compress size of the  elements when skipping \*/  intn skp\_size;  } skphuff;  struct  { /\* struct to contain info about how to compress or  decompress gzip encoded dataset how hard to work  when compressing data\*/  intn level;  } deflate;  struct  {  int32 options\_mask; /\* IN \*/  int32 pixels\_per\_block; /\* IN \*/  int32 pixels\_per\_scanline; /\* OUT: computed \*/  int32 bits\_per\_pixel; /\* OUT: size of NT \*/  int32 pixels; /\* OUT: size of dataset or chunk \*/  } szip; /\* for szip encoding \*/  } comp\_info; | | |
| FORTRAN | integer mgscompress(ri\_id, comp\_type, comp\_prm) | | |
|  | integer ri\_id, comp\_type, comp\_prm(\*) | | |

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 GRwriteimage. | | |
|  | 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 | | |

GRwritechunk/mgwchnk/mgwcchnk

intn GRwritechunk(int32 ri\_id, int32 \*origin, const VOIDP datap)

|  |  |  |
| --- | --- | --- |
| *ri\_id* | IN: | Raster image identifier returned by GRcreate or GRselect |
| *origin* | IN: | Origin of the chunk to be written |
| *datap* | IN: | Buffer for the chunk to be written |
| Purpose | Writes a data chunk to a chunked raster image (pixel-interlace only) | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
|  | GRwritechunk returns FAIL (or -1) when an attempt is made to use it to write to a non-chunked raster image. | | |
| Description | 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 parameter origin is a two-dimensional array which specifies the coordinates of the chunk according to the chunk position in the overall chunk array. Refer to Chapter 8, "General Raster Images (GR API)" in the HDF User’s Guide. | | |
|  | The datap buffer contains the chunk’s data organized in a pixel interlace mode. | | |
| FORTRAN | integer mgwchnk(ri\_id, origin, datap) | | |
|  | integer ri\_id, origin(2) | | |
|  | <valid\_numeric\_datatype> datap(\*) | | |
|  |  | | |
|  | integer mgwcchnk(ri\_id, origin, char\_datap) | | |
|  | integer ri\_id, origin(2) | | |
|  | character\*(\*) char\_datap | | |

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. | | |
|  | 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. | | |
|  | Any un-written elements in the image will be filled with the image’s fill value. If the fill-value is previously set, it will be used, otherwise, the default fill-value, which is 0, will be used. | | |
|  | Note that there are two FORTRAN-77 versions of this routine; one for numeric data (mgwrimg) and the other for character data (mgwcimg). | | |
| Note | **Regarding an important difference between the SD and GR interfaces:** 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 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.  o 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.  o 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. | | |
| 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 | | |

GR2bmapped

int32 GR2bmapped(int32 *ri\_id*, intn \**tobe\_mapped*, intn \**name\_generated*)

|  |  |  |
| --- | --- | --- |
| ri\_id | IN: | Raster image identifier returned by GRselect |
| tobe\_mapped | OUT: | TRUE if the image should be mapped |
| name\_generated | OUT: | TRUE if the image’s name was generated by the library |
| Purpose | Checks whether a raster image is to be mapped | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | GR2bmapped determines whether the given raster image satisfies the following conditions:  **o** being an 8-bit raster image  **o** having one component  **o** being non-special or RLE compressed only, i.e., no other  compressions, no chunking,...  The function will set tobe\_mapped to TRUE if the image satisfies the above conditions, and FALSE, otherwise. | | |
|  | In addition, GR2bmapped will set the flag name\_generated to indicate whether the image has name that was generated by the library instead of given by application. 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 HDF4 File Content Writer needs to make this distinction. | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |