SDattrinfo/sfgainfo

intn SDattrinfo(int32 *obj\_id,* int32 *attr\_index*, char \**attr\_name*, int32 \**ntype*, int32 \**count*)

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Identifier of the object to which the attribute is attached to |
| attr\_index | IN: | Index of the attribute |
| attr\_name | OUT: | Name of the attribute |
| ntype | OUT: | Number type of the attribute values |
| count | OUT: | Total number of values in the attribute |
| Purpose | Retrieves information about an attribute. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDattrinfo retrieves the name, number type, and number of values of the attribute specified by its index, attr\_index, and stores them in the parameters attr\_name, ntype, and count, respectively. This routine should be used before reading the values of an attribute with SDreadattr. | | |
|  | The parameter obj\_id can be either an SD interface identifier (sd\_id), returned by SDstart, a data set identifier (sds\_id), returned by SDselect, or a dimension identifier (dim\_id), returned by SDgetdimid. | | |
|  | Valid values of the parameter attr\_index range from 0 to the number of attributes attached to the object - 1. | | |
|  | Valid values of the parameter ntype can be found in Table 1A of Section I in this manual. | | |
| FORTRAN | integer function sfgainfo(obj\_id, attr\_index, attr\_name, ntype, count) | | |
|  | character\*(\*) attr\_name | | |
|  | integer obj\_id, attr\_index, ntype, count | | |

SDcheckempty/sfchempty

int32 SDcheckempty( int32 *sds\_id*, intn \**emptySDS* )

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | SDS identifier |
| emptySDS | OUT: | Boolean value indicating whether the SDS is empty |
| Purpose | Determines whether an SDS is empty. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDcheckempty sets the parameter emptySDS to TRUE if the dataset identified by sds\_id has not been written with data, and to FALSE, otherwise. | | |
|  | The Fortran routine, sfchempty, returns 1 in emptySDS if the dataset is empty and 0 otherwise. | | |
| FORTRAN | integer function sfchempty(sds\_id, emptySDS) | | |
|  | integer sds\_id, emptySDS | | |

SDcreate/sfcreate

int32 SDcreate(int32 *sd\_id*, char \**name*, int32 *ntype*, int32 *rank*, int32 *dimsizes*[])

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by SDstart |
| name | IN: | Name of the data set |
| ntype | IN: | Number type for the values in the data set |
| rank | IN: | Number of the data set dimensions |
| dimsizes | IN: | Array containing the size of each dimension |
| Purpose | Creates a new data set. | | |
| Return value | Returns the data set identifier (sds\_id) if successful and FAIL (or -1) otherwise. | | |
| Description | SDcreate creates a data set with the name, number type, number of dimensions, dimension sizes specified by the parameters name, ntype, rank, and dimsizes. | | |
|  | Once a data set has been created, it is not possible to change its name, number type, or rank. However, it is possible to create a data set and close the file before writing any data values to it. The values can be added or modified at a future time. To add data or modify an existing data set, use SDselect to get the data set identifier instead of SDcreate. | | |
|  | If the parameter name is NULL in C or an empty string in Fortran, the default name “DataSet” will be generated. The length of the name specified by the name parameter is no longer limited to 64 characters starting in HDF 4.2r2. Note that when an older version of the library reads a data set, which was created by a library of version 4.2r2 or later and has the name that is longer than 64 characters, the retrieved name will contain some garbage after 64 characters. | | |
|  | The calling program must ensure that the length of the dimsizes array is the value of the rank parameter, which is between 0 and MAX\_VAR\_DIMS (or 32). Note that, in order for HDF4 and NetCDF models to work together, HDF allows SDS to have rank 0. However, there is no intention for data to be written to this type of SDS, but only to store attribute as part of the data description. Consequently, setting compression and setting chunk are disallowed. | | |
|  | To create a data set with an unlimited dimension, assign the value of SD\_UNLIMITED (or 0) to dimsizes[0] in C and to dimsizes(rank) in Fortran. | | |
|  | The ntype parameter can contain any number type supported by the HDF library. These number types are listed in Table 1A, Number Type Definitions of this manual. | | |
|  | See the notes regarding the potential performance impact of unlimited dimension data sets in Section 14.4.3, "Unlimited Dimension Data Sets (SDSs and Vdatas) and Performance" the HDF User’s Guide. | | |
| 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 sfcreate(sd\_id, name, ntype, rank, dimsizes) | | |
|  | character\*(\*) name | | |
|  | integer sd\_id, ntype, rank, dimsizes(\*) | | |

SDdiminfo/sfgdinfo

intn SDdiminfo(int32 *dim\_id*, char \**name*, int32 \**size*, int32 \**ntype*, int32 \**num\_attrs*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| name | OUT: | Name of the dimension |
| size | OUT: | Size of the dimension |
| ntype | OUT: | Number type of the dimension scale |
| num\_attrs | OUT: | Number of attributes assigned to the dimension |
| Purpose | Retrieves information about a dimension. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDdiminfo retrieves the name, size, number type, and number of values of the dimension specified by the parameter dim\_id, and stores them in the parameters name, size, ntype, and num\_attrs, respectively. | | |
|  | If the output value of the parameter size is set to 0, then the dimension specified by the dim\_id parameter is unlimited. To get the number of records of an unlimited dimension, use SDgetinfo. | | |
|  | If scale information has been stored for this dimension via SDsetdimscale, the ntype parameter will contain the number type. Valid number types can be found in Table 1A, Number Type Definitions, in this manual. If no scale information has been stored for this dimension, the value returned in the ntype parameter will be 0. | | |
|  | If the user has not named the dimension via SDsetdimname, a default dimension name of “fakeDim[x]” will be generated by the library, where [x] denotes the dimension index. If the name is not desired, the parameter name can be set to NULL in C and an empty string in Fortran. | | |
| FORTRAN | integer function sfgdinfo(dim\_id, name, size, ntype, num\_attrs) | | |
|  | character\*(\*) name | | |
|  | integer dim\_id, size, ntype, num\_attrs | | |

SDend/sfend

intn SDend(int32 *sd\_id*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by SDstart |
| Purpose | Terminates access to an SD interface. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDend closes the file and frees memory allocated by the library when SD interface activities are completed. If the calling program exits without invoking this routine, recent changes made to the in-core file data are likely not to be flushed to the file. Note that each SDstart must have a matching SDend. | | |
| FORTRAN | integer function sfend(sd\_id) | | |
|  | integer sd\_id | | |

SDendaccess/sfendacc

intn SDendaccess(int32 *sds\_id*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| Purpose | Terminates access to a data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDendaccess frees the memory taken up by the HDF library’s data structures devoted to the data set identified by the parameter sds\_id. | | |
|  | Failing to call this routine after all operations on the specified data set are complete may result in loss of data. This routine must be called once for each call to SDcreate or SDselect. | | |
| FORTRAN | integer function sfendacc(sds\_id) | | |
|  | integer sds\_id | | |

SDfileinfo/sffinfo

intn SDfileinfo(int32 *sd\_id*, int32 \**num\_datasets*, int32 \**num\_global\_attrs*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by SDstart |
| num\_datasets | OUT: | Number of data sets in the file |
| num\_global\_attrs | OUT: | Number of global attributes in the file |
| Purpose | Retrieves the number of data sets and the number of global attributes in a file. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDfileinfo returns the number of data sets in the parameter num\_datasets and the number of global attributes in the parameter num\_global\_attrs. The term “global attributes” refers to attributes that are assigned to the file. The global attributes are created by SDsetattr using an SD interface identifier (sd\_id) rather than a data set identifier (sds\_id). | | |
|  | The value returned by the parameter num\_datasets includes the number of coordinate variable data sets. To determine if the data set is a coordinate variable, use SDiscoordvar. | | |
| FORTRAN | integer function sffinfo(sd\_id, num\_datasets, num\_global\_attrs) | | |
|  | integer sd\_id, num\_datasets, num\_global\_attrs | | |

SDfindattr/sffattr

int32 SDfindattr(int32 *obj\_id*, char \**attr\_name*)

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Identifier of the object to which the attribute is attached |
| attr\_name | IN: | Name of the attribute |
| Purpose | Finds the index of an attribute given its name. | | |
| Return value | Returns the index if successful and FAIL (or -1) otherwise. | | |
| Description | **SDfindattr** retrieves the index of the object’s attribute with the name specified by the parameter attr\_name. | | |
|  | The attribute is attached to the object specified by the parameter obj\_id. The parameter obj\_id can be either an SD interface identifier (sd\_id), returned by **SDstart**, a data set identifier (sds\_id), returned by **SDselect**, or a dimension identifier (dim\_id), returned by **SDgetdimid**. | | |
|  | Wildcard characters are not allowed in the parameter attr\_name. **SDfindattr** searches for the name specified in the parameter attr\_name in a case-sensitive manner. | | |
| FORTRAN | integer function sffattr(obj\_id, attr\_name) | | |
|  | integer obj\_id | | |
|  | character\*(\*) attr\_name | | |

SDgetanndatainfo

intn SDgetanndatainfo(int32 *sds\_id*, ann\_type *annot\_type,* uintn *info\_count*, int32 *\*offsetarray,* int32 *\*lengtharray*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDselect** |
| annot\_type | IN: | Type of annotations to retrieve data information |
| info\_count | IN: | Number of elements in offsetarray and lengtharray |
| offsetarray | OUT: | Buffer to hold offsets of the annotations’ data |
| lengtharray | OUT: | Buffer to hold lengths of the annotations’ data |
| Purpose | Retrieves location and size of annotations’ data. | | |
| Return value | Returns the number of annotation data information retrieved, if successful, and FAIL (or -1) otherwise. | | |
| Description | SDgetanndatainfo retrieves the location and size specifying the data of annotations that are of the specific type, annot\_type, and are assigned to the SDS sds\_id. There may be more than one annotation, but each annotation has only one block of data. | | |
|  | The type annot\_type can be one of the following values: AN\_DATA\_LABEL (0), AN\_DATA\_DESC (1), AN\_FILE\_LABEL (2), AN\_FILE\_DESC (3.) | | |
|  | 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 SDgetanndatainfo passing in 0 for info\_count and NULL for both arrays to get the value for info\_count in the next call to SDgetanndatainfo. | | |
| Note | If the caller provides buffers that are smaller than the number of annotations then SDgetanndatainfo only fills the buffers up to its size, starting from the first annotation. That means, the rest of the annotations are not retrievable. Thus, obtaining info\_count to sufficiently allocate the buffers is recommended. | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |

SDgetattdatainfo

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

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Identifier of the object the attribute is attached to |
| 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, or  - DFE\_NOVGREP if the attribute is the old style (created by DFSD API,) or  - FAIL (or -1) if failure occurs. | | |
| Description | SDgetattdatainfo 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 an SD interface identifier (sd\_id), returned by SDstart, a data set identifier (sds\_id), returned by SDselect, or a dimension identifier (dim\_id), returned by SDgetdimid. | | |
|  | There are attributes created by SDsetattr and those created by the DFSD API functions. SDgetattdatainfo can only retrieve data information of attributes that were created by SDsetattr. If the inquired attribute was created by the DFSD API functions, SDgetattdatainfo will return to the caller with error code DFE\_NOVGREP so the caller can call SDgetoldattdatainfo to get the attribute’s data information. | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |

SDgetcal/sfgcal

intn SDgetcal(int32 *sds\_id*, float64 \**cal*, float64 \**cal\_err*, float64 \**offset*, float64 \**offset\_err*, int32 \**ntype*)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sds\_id | IN: | | Data set identifier returned by **SDcreate** or **SDselect** | | |
| cal | OUT: | | Calibration factor | | |
| cal\_err | OUT: | | Calibration error | | |
| offset | OUT: | | Uncalibrated offset | | |
| offset\_err | OUT: | | Uncalibrated offset error | | |
| ntype | OUT: | | Number type of uncalibrated data | | |
| Purpose | Retrieves the calibration information associated with a data set. | | | | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | | | | |
| Description | **SDgetcal** reads the calibration record attached to the data set identified by the parameter sds\_id. A calibration record is comprised of four 64-bit floating point values followed by a 32-bit integer. The information is listed in the following table: | | | | | |
| cal | | calibration factor |
| cal\_err | | calibration error |
| offset | | uncalibrated offset |
| offset\_err | | uncalibrated offset error |
| ntype | | number type of the uncalibrated data |
|  | The relationship between a calibrated value cal\_value and the original value orig\_value is defined as orig\_value = cal \* (cal\_value - offset). | | | | | |
|  | The variable offset\_err contains a potential error of offset, and cal\_err contains a potential error of cal. Currently the calibration record is provided for information only. The SD interface performs no operations on the data based on the calibration tag. | | | | | |
| FORTRAN | integer function sfgcal(sds\_id, cal, cal\_err, offset, offset\_err, ntype) | | | | | |
|  | integer sds\_id, ntype | | | | | |
|  | real\*8 cal, cal\_err, offset, offset\_err | | | | | |

SDgetchunkinfo/sfgichnk

intn SDgetchunkinfo(int32 *sds\_id*, HDF\_CHUNK\_DEF \**cdef*, int32 \**flag*)

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| sds\_id | IN: | | Data set identifier returned by **SDcreate** or **SDselect** | | | |
|  |  | |  | | | |
| *C only:* |  | |  | | | |
| cdef | OUT: | | Pointer to the chunk definition | | | |
| flag | OUT: | | Compression flag | | | |
|  |  | |  | | | |
| *Fortran only:* |  | |  | | | |
| dim\_length | OUT: | | Array of chunk dimensions | | | |
| flag | OUT: | | Compression flag | | | |
| Purpose | Retrieves chunking information for a data set. | | | | | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | | | | | |
| Description | **SDgetchunkinfo** retrieves chunking information about the data set identified by the parameter sds\_id into the parameters cdef and flag in C, and to the parameters dim\_length and flag in Fortran. | | | | | | |
|  | Currently, only information about chunk dimensions is retrieved into the corresponding cdef structure element for each type of compression in C, and in the dim\_length array in Fortran. No information on compression parameters is available in the comp structure of the HDF\_CHUNK\_DEF union. Refer to the page on **SDsetchunk** in this manual for specific information on the HDF\_CHUNK\_DEF union. | | | | | | |
|  | The value returned in the flag parameter indicates the data set type (i.e., if the data set is not chunked, chunked, and chunked and compressed). | | | | | | |
|  | If the chunk length for each dimension is not needed, NULL can be passed in as the value of the cdef parameter in C. | | | | | | |
|  | The following table shows the type of the data set, possible values of the flag parameter, and the corresponding cdef structure element filled with the chunk’s dimensions. | | | | | | |
| Type of Data Set | | Values of *flag* in C | Values of *flag* in Fortran | *cdef* Structure Element Filled with the Chunk’s Dimensions | |
| Not chunked | | HDF\_NONE | -1 | None | |
| Chunked | | HDF\_CHUNK | 0 | cdef.chunk\_lengths[] | |
| Chunked and compressed with either the run-length encoding (RLE), Skipping Huffman, GZIP, or Szip compression algorithms | | HDF\_CHUNK | HDF\_COMP | 1 | cdef.comp.chunk\_lengths[] | |
| Chunked and compressed with NBIT compression | | HDF\_CHUNK | HDF\_NBIT | 2 | cdef.nbit.chunk\_lengths[] | |

|  |  |
| --- | --- |
| FORTRAN | integer function sfgichnk(sds\_id, dim\_length, flag) |
|  | integer sds\_id, dim\_length(\*), flag |

SDgetcompinfo/sfgcompress

intn SDgetcompinfo(int32 *sds\_id*, comp\_coder\_t \**comp\_type*, comp\_info \**c\_info*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| comp\_type | OUT: | Type of compression |
| c\_info | OUT: | Pointer to compression information structure |
| Purpose | Retrieves data set compression type and compression information. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetcompinfo** retrieves the compression type and compression information for a data set, when the data is either compressed, chunked or chunked and compressed. SDgetcompinfo replaces SDgetcompress 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 --for no compression  COMP\_CODE\_RLE --for RLE run-length encoding  COMP\_CODE\_NBIT --for NBIT compression  COMP\_CODE\_SKPHUFF -for Skipping Huffman compression  COMP\_CODE\_DEFLATE --for-GZIP compression  COMP\_CODE\_SZIP --for Szip compression | | |
|  | Additional compression method parameters are returned in the c\_info struct in C and the array parameter comp\_prm in Fortran. Note that c\_info and comp\_prm come into place only with compression modes that require additional parameters (i.e., other than comp\_type); they are ignored in other cases. | | |
|  | The c\_info struct is of type comp\_info, contains algorithm-specific information for the library compression routines, and is described in the SDsetcompress entry in this reference manual and in the hcomp.h header file. | | |
|  | The comp\_prm parameter is an array returning one or more parameters, as required by the compression method in use. Each compression parameter is returned as an element of the array, as follows:  o With Skipping Huffman compression, comp\_prm is a 1-element array and comp\_prm(1) contains the skip value, skphuff\_skp\_size.  o In the case of GZIP compression, comp\_prm is also a 1-element array and comp\_prm(1) contains the deflation value, deflate\_value.  o In the case of NBIT compression, comp\_prm is a 4-element array with sign\_ext in comp\_prm(1), fill\_one in comp\_prm(2), start\_bit in comp\_prm(3), and bit\_len in comp\_prm(4). The fields sign\_ext, fill\_one, start\_bit, and bit\_len are discussed in the SDsetnbitdataset/sfsnbit entry of this reference manual.  o In the case of Szip compression, comp\_prm is a 5-element array with option\_mask in comp\_prm(1), pixels\_per\_block in comp\_prm(2), pixels\_per\_scanline in comp\_prm(3), bits\_per\_pixel in comp\_prm(4), and pixels in comp\_prm (5). | | |
|  | In the general case, any available compression type can be configured in any mode:  COMP\_DECODER\_ENABLED Decode data only  COMP\_ENCODER\_ENABLED Encode data only  COMP\_DECODER\_ENABLED |COMP\_ENCODER\_ENABLED   Decode and encode data  As of this writing (HDF4 Release 2.1, February 2005), only the Szip compression library is actually used with the HDF libraries in more than one configuration (see immediately below). As a third-party product, it is distributed in both decode-only and encode/decode configurations. All other compression methods are currently distributed or used in an encode/decode configuration if they are available at all. See also HCget\_config\_info. | | |
|  | **SDgetcompinfo** will succeed for an Szip-compressed dataset whether the available Szip library is configured either for encoding/decoding or for decoding-only.  If the Szip configuration is decode-only, i.e., an **HCget\_config\_info** call on the dataset will return only COMP\_DECODER\_ENABLED in compression\_config\_info. Note that in such a case the file must be opened in read-only mode, i.e. with SDstart(filename, DFACC\_RDONLY).  If the Szip configuration is encode/decode, i.e., an **HCget\_config\_info** call on the dataset will return COMP\_ENCODER\_ENABLED|COMP\_DECODER\_ENABLED in compression\_config\_info. In this case, the file and dataset can be opened in read/write mode. | | |
| Note | **Regarding uncompressed data or an empty data set:**  **SDgetcompinfo** will succeed and the parameter comp\_type will have the value COMP\_CODE\_NONE if either of the following conditions exists:  o The data set is not compressed.  o No data has been written to the SDS. | | |
| Note | **Regarding Szip usage and licensing:**  See http://www.hdfgroup.org/doc\_resource/SZIP/ for information regarding the use of Szip in HDF products and Szip licensing. | | |
| FORTRAN | integer function sfgcompress(sds\_id, comp\_type, comp\_prm) | | |
|  | integer sds\_id, comp\_type, comp\_prm(\*) | | |
|  |  | | |

SDgetdatainfo

intn SDgetdatainfo(int32 *sds\_id*, int32 \**chk\_coord*, uintn *start\_block*, uintn info\_count, int32 \**offsetarray*, int32 *\*lengtharray*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | SDS identifier returned by SDselect |
| chk\_coord | IN: | Chunk coord array or NULL for non-chunk SDS |
| 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 data set, starting at a specified block. | | |
| Return value | Returns the number of data blocks retrieved if successful, and FAIL (or -1) otherwise. | | |
| Description | SDgetdatainfo retrieves two lists, offsetarray and lengtharray, containing the offsets and lengths of the blocks of data belonging to the data set specified by sds\_id. | | |
|  | The parameter info\_count provides the number of offset/length values that the lists can hold. The application can first invoke SDgetdatainfo passing in 0 for info\_count and NULL for both arrays to get the value for info\_count and to provide proper memory allocation for offsetarray and lengtharray in the next call to SDgetdatainfo. | | |
|  | The parameter start\_block indicates where to start reading the offsets from in the file. The combination of parameters info\_length and start\_block provide user applications with flexibility of where and how much data information to retrieve. The value for start\_block must be non-negative and smaller than or equal to the number of blocks in the data set. | | |
|  | o When start\_block is 0, SDgetdatainfo will start getting data info from the beginning of the data set's data.  o When start\_block is greater than the number of blocks in the data set, SDgetdatainfo will return FAIL (or -1). | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |

SDgetdatastrs/sfgdtstr

intn SDgetdatastrs(int32 *sds\_id*, char \**label*, char \**unit*, char \**format*, char \**coordsys*, intn *length*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| label | OUT: | Label (predefined attribute) |
| unit | OUT: | Unit (predefined attribute) |
| format | OUT: | Format (predefined attribute) |
| coordsys | OUT: | Coordinate system (predefined attribute) |
| length | IN: | Maximum length of the above predefined attributes |
| Purpose | Retrieves the predefined attributes of a data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetdatastrs** retrieves the predefined attributes for the data set specified by the parameter sds\_id. The predefined attributes are label, unit, format, and coordinate system. They are then stored in the parameters label, unit, format, and coordsys, respectively. Refer to Section 3.10, "Predefined Attributes" of the HDF User’s Guide for more information on predefined attributes. | | |
|  | If a particular data string is not stored, the first character of the corresponding **SDgetdatastrs** parameter is '\0' in C. In FORTRAN, the parameter contains an empty string. Each string buffer must include the space to hold the null termination character. In C, if a user does not want a string back, NULL can be passed in for that string. Data strings are set by the **SDsetdatastrs** routine. | | |
| FORTRAN | integer function sfgdtstr(sds\_id, label, unit, format, coordsys, length) | | |
|  | integer sds\_id, length | | |
|  | character\*(\*) label, unit, format, coordsys | | |

SDgetdimid/sfdimid

int32 SDgetdimid(int32 *sds\_id*, intn *dim\_index*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| dim\_index | IN: | Index of the dimension |
| Purpose | Returns the identifier of a dimension given its index. | | |
| Return value | Returns the dimension identifier (dim\_id) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetdimid** returns the identifier of the dimension specified by its index, the parameter dim\_index. | | |
|  | The dimension index is a nonnegative integer and is less than the total number of data set dimensions returned by **SDgetinfo**. | | |
| FORTRAN | integer function sfdimid(sds\_id, dim\_index) | |
|  | integer sds\_id, dim\_index | |

SDgetdimscale/sfgdscale

intn SDgetdimscale(int32 *dim\_id*, VOIDP *scale\_buf*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| scale\_buf | OUT: | Buffer for the scale values |
| Purpose | Retrieves the scale values for a dimension. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetdimscale** retrieves the scale values of the dimension identified by the parameter dim\_id and stores the values in the buffer scale\_buf. | | |
|  | Prior to calling SDgetdimscale, the application should use **SDdiminfo** to determine whether a scale had been set for the dimension, i.e., that the dimension scale’s number type is a valid HDF type, as listed in Table 1A, Number Type Definitions, not 0. If there is no scale, the buffer returned by SDgetdimscale is meaningless. **SDdiminfo** also provides the number of scale values for space allocation before passing the buffer into **SDgetdimscale**. | | |
|  | It is not possible to read a subset of the scale values. **SDgetdimscale** returns all of the scale values stored with the given dimension. | | |
| FORTRAN | integer function sfgdscale(dim\_id, scale\_buf) | | |
|  | integer dim\_id | | |
|  | <valid numeric data type> scale\_buf(\*) | | |

SDgetdimstrs/sfgdmstr

intn SDgetdimstrs(int32 *dim\_id*, char \**label*, char \**unit*, char \**format*, intn *length*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| label | OUT: | Label (predefined attribute) |
| unit | OUT: | Unit (predefined attribute) |
| format | OUT: | Format (predefined attribute) |
| length | IN: | Maximum length of the above predefined attributes |
| Purpose | Retrieves the predefined attributes of a dimension. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetdimstrs** retrieves the predefined attributes associated with the dimension identified by the parameter dim\_id. The predefined attributes are label, unit, and format. These predefined attributes are stored in the parameters label, unit, and format, respectively. Refer to Table 3.10, Predefined Attributes, in the HDF User’s Guide for more information on predefined attributes. | | |
|  | If a particular data string was not stored, the first character of the corresponding **SDgetdimstrs** parameter is '\0'. Each string buffer must include space for the null termination character. If a user does not want a string returned, the corresponding parameter can be set to NULL in C and an empty string in Fortran. The predefined attributes are set by **SDsetdimstrs**. | | |
| FORTRAN | integer function sfgdmstr(dim\_id, label, unit, format, length) | | |
|  | integer dim\_id, length | | |
|  | character\*(\*) label, unit, format | | |

SDgetexternalinfo

intn SDgetexternalinfo(int32 *sds\_id,* uintn *buf\_size*, char \**filename*, int32 \**offset*, int32 \**length*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| buf\_size | IN: | Size of buffer for external file name |
| filename | OUT: | Buffer for external file name |
| offset | OUT: | Offset, in bytes, of the location in the external file where the data was written |
| length | OUT: | Length, in bytes, of the external data |
| Purpose | Retrieves information about external file and external data of the data set. | | |
| Return value | Returns length of the external file name if successful, 0 if there is no external data, or or FAIL (or -1) if an error occurs. | | |
| Description | If the data set has external element, SDgetexternalinfo will retrieve the name of the external file, the offset where the data is being stored in the external file, and the length of the external data. If the data set does not have external element, SDgetexternalinfo will return 0. | | |
|  | To sufficiently allocate buffer for the file name, an application can call SDgetexternalinfo passing in 0 for buf\_size. If the length returned is greater than 0, the application will use it to allocate the buffer before calling SDgetexternalinfo again to get the actual file name. | | |
| Note | It is the user's responsibility to see that the external files are kept with the main file prior to accessing the data set with external element. SDgetexternalinfo does not check and the accessing functions will fail if the external file is missing from the directory where the main file is located. | | |
| FORTRAN | Currently unavailable | | |
|  |  | | |

SDgetfilename

intn SDgetfilename(int32 file\_id, char \*filename)

|  |  |  |
| --- | --- | --- |
| file\_id | IN: | A file identifier |
| filename | OUT: | Name of the file |
| Purpose | Given a file identifier, retrieves the name of the file. | | |
| Return value | Returns the length of the file name, without '\0', on success, and FAIL, otherwise. | | |
| FORTRAN | integer function sfgetfname(file\_id, filename) | | |
|  | integer file\_id | | |
|  | character\*(\*) filename | | |

SDgetfillvalue/sfgfill/sfgcfill

intn SDgetfillvalue(int32 *sds\_id*, VOIDP *fill\_value*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| fill\_value | OUT: | Buffer for the returned fill value |
| Purpose | Reads the fill value of a data set, if the value has been set. | | |
| Return value | Returns SUCCEED (or 0) if a fill value is retrieved and FAIL (or -1) otherwise, including when the fill value is not set. | | |
| Description | SDgetfillvalue reads the fill value which has been set for the data set specified by the parameter sds\_id. It is assumed that the type of the fill value is the same as that of the data set. | | |
|  | The following are the default fill values for different types:  FILL\_BYTE (char)-127 /\* Largest Negative value \*/  FILL\_CHAR (char)0  FILL\_SHORT (short)-32767  FILL\_LONG (long)-2147483647  FILL\_FLOAT 9.9692099683868690e+36 /\* near 15 \* 2^119 \*/  FILL\_DOUBLE 9.9692099683868690e+36 | | |
|  | Note that there are two FORTRAN-77 versions of this routine: sfgfill and sfgcfill. The sfgfill routine reads numeric fill value data and sfgcfill reads character fill value data. | | |
| FORTRAN | integer function sfgfill(sds\_id, fill\_value) | | |
|  | integer sds\_id | | |
|  | <valid numeric data type> fill\_value | | |
|  |  | | |
|  | integer function sfgcfill(sds\_id, fill\_value) | | |
|  | integer sds\_id | | |
|  | character\*(\*) fill\_value | | |

SDgetinfo/sfginfo

intn SDgetinfo(int32 *sds\_id*, char \**sds\_name*, int32 \**rank*, int32 *dimsizes*[], int32 \**ntype*, int32 \**num\_attrs*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** and **SDselect** |
| sds\_name | OUT: | Name of the data set |
| rank | OUT: | Number of dimensions in the data set |
| dimsizes | OUT: | Array containing the size of each dimension in the data set |
| ntype | OUT: | Number type for the data stored in the data set |
| num\_attrs | OUT: | Number of attributes for the data set |
| Purpose | Retrieves the name, rank, dimension sizes, number type and number of attributes for a data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetinfo** retrieves the name, number of dimensions, sizes of dimensions, number type, and number of attributes of the data set identified by sds\_id, and stores them in the parameters sds\_name, rank, dimsizes, ntype, and num\_attrs, respectively. | | |
|  | The buffer sds\_name must be sufficiently allocated. The application may call SDgetnamelen to determine the needed space. If the name of the data set is not desired, then the parameter sds\_name can be set to NULL in C and an empty string in Fortran. | | |
|  | The maximum value for rank is MAX\_VAR\_DIMS (or 32.) | | |
|  | If the data set is created with an unlimited dimension, then in the C interface, the first element of the dimsizes array (corresponding to the slowest-changing dimension) contains the number of records in the unlimited dimension; in the FORTRAN-77 interface, the last element of the dimsizes array (corresponding to the slowest-changing dimension) contains this information. Use **SDisrecord** to determine if the data set has an unlimited dimension. | | |
| FORTRAN | integer function sfginfo(sds\_id, sds\_name, rank, dimsizes, ntype, num\_attrs) | | |
|  | character\*(\*) sds\_name | | |
|  | integer sds\_id, rank, dimsizes(\*) | | |
|  | integer ntype, num\_attrs | | |

SDgetnamelen

intn SDgetnamelen(int32 obj\_id, uint16 name\_len)

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Identifier of the object |
| name\_len | OUT: | Length of the object’s name |
| Purpose | Retrieves the length of the name of a file, a dataset, or a dimension. | | |
| Return value | Returns the length of the object’s name on success, and FAIL (or -1), otherwise. | | |
| Description | Given an identifier of a file, a dataset, or a dimension,SDgetnamelen retrieves the length of its name into name\_len. The length does not include the character '\0'. | | |
| FORTRAN | integer function sfgetnamelen(obj\_id, length) | | |
|  | integer obj\_id, length | | |

SDgetnumvars\_byname

intn SDgetnumvars\_byname(int32 *sd\_id*, char \**sds\_name,* unsigned *\*n\_vars*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by **SDstart** |
| sds\_name | IN: | Name of the data set |
| n\_vars | OUT: | Number of variables named sds\_name |
| Purpose | Get the number of data sets having the same name. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetnumvars\_byname** retrieves the number of variables with the name specified by the parameter sds\_name. The variables may include both data sets or coordinate variables. The routine does not accept wildcards in the specified data set name. It also searches on that name in a case-sensitive manner. | | |
| FORTRAN | integer function sfgnvars\_byname(sd\_id, sds\_name, n\_vars) | | |
|  | integer sd\_id, n\_vars | | |
|  | character\*(\*) sds\_name | | |

SDgetoldattdatainfo

intn SDgetoldattdatainfo(int32 *dim\_id*, int32 *sds\_id*, char \**attr\_name,* int32 *\*offset,* int32 *\*length*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by SDgetdimid |
| sds\_id | IN: | SDS identifier returned by SDselect |
| attr\_name | IN: | Name 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 old predefined attribute's data. |
| Return value | Returns number of data blocks retrieved, which should be 1 if successful and FAIL (or -1) otherwise. |
| Description | SDgetoldattdatainfo retrieves the offset and length of the data that belongs to the attribute attr\_name, which is attached to the SDS sds\_id or the dimension dim\_id. |
|  | The function only works on attributes that were created by the DFSD API while its counter part SDgetattdatainfo only works on attributes created with SDsetattr. An application might call SDgetattdatainfo initially. When a DFSD-created attribute is encountered, SDgetattdatainfo will fail with the error code DFE\_NOVGREP, which indicates there is no vgroup representation for an SDS (i.e., DFSD API) and the SDS' attributes are stored differently than when they are created with SDsetattr. The application must call SDgetoldattdatainfo to get the data information of those attributes, if such error code is detected. |
|  | SDgetoldattdatainfo takes both SDS identifier and dimension identifier if the inquired attribute belongs to a dimension. When the inquired attribute belongs to an SDS, the dimension identifier will not be needed, and should be 0. |
|  | The attribute is a predefined attribute listed in the following table and is passed in for attr\_name. Note that, dimensions can only have the first three attributes in the table. |

|  |  |  |
| --- | --- | --- |
| HDF4 Predefined Attributes | | |
| ****Predefined Name**** | ****Actual Text**** | ****Applicable To**** |
| \_HDF\_LongName | "long\_name" | Dimension & SDS |
| \_HDF\_Units | "units" | Dimension & SDS |
| \_HDF\_Format | "format" | Dimension & SDS |
| \_HDF\_CoordSys | "coordsys" | Only SDS |
| \_HDF\_ScaleFactorErr | "scale\_factor\_err" | Only SDS |
| \_HDF\_AddOffset | "add\_offset" | Only SDS |
| \_HDF\_ValidRange | "valid\_range" | Only SDS |
| \_HDF\_ScaleFactor | "scale\_factor" | Only SDS |
| \_HDF\_AddOffsetErr | "add\_offset\_err" | Only SDS |
| \_HDF\_CalibratedNt | "calibrated\_nt" | Only SDS |
| \_HDF\_ValidMax | "valid\_max" | Only SDS |
| \_HDF\_ValidMin | "valid\_min" | Only SDS |
| \_FillValue | "\_FillValue" | Only SDS |

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

SDgetrange/sfgrange

intn SDgetrange(int32 *sds\_id*, VOIDP *max*, VOIDP *min*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| max | OUT: | Maximum value of the range |
| min | OUT: | Minimum value of the range |
| Purpose | Retrieves the maximum and minimum values of the range. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDgetrange** retrieves the maximum value of the range into the parameter max and the minimum value into the parameter min. The maximum and minimum values must be previously set via a call to **SDsetrange**. | | |
|  | It is assumed that the number type for the maximum and minimum range values are the same as that of the data. | | |
| FORTRAN | integer function sfgrange(sds\_id, max, min) | | |
|  | integer sds\_id | | |
|  | <valid numeric data type> max, min | | |

SDget\_maxopenfiles

intn SDget\_maxopenfiles(intn \*curr\_max, intn \*sys\_limit)

|  |  |  |
| --- | --- | --- |
| cu | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| curr\_max | OUT: | Current number of open files |
| sys\_limit | OUT: | Maximum number of open files |
| Purpose | Retrieves current and maximum number of open files. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDget\_maxopenfiles retrieves the current number of open files allowed in HDF, curr\_max, and the maxinum number of open files allowed on the system, sys\_limit. If either of the values is not desired, then NULL can be passed in. | | |
| FORTRAN | integer function sfgmaxopenf(cur\_max, sys\_limit) | | |
|  | integer cur\_max, sys\_limit | | |

SDget\_numopenfiles

intn SDget\_numopenfiles()

|  |  |
| --- | --- |
| Purpose | Returns the number of files currently being opened. |
| Return value | Returns the number of files currently being opened. |
| FORTRAN | integer function sfgnumopenf(cur\_num) |
|  | integer cur\_num |

SDidtoref/sfid2ref

int32 SDidtoref(int32 *sds\_id*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| Purpose | Returns the reference number assigned to a data set. | | |
| Return value | Returns the data set reference numberif successful and FAIL (or -1) otherwise. | | |
| Description | **SDidtoref** returns the reference number of the data set specified by the parameter sds\_id. The reference number is assigned by the HDF library when the data set is created. The specified reference number can be used to add the data set to a vgroup as well as a means of using the HDF annotations interface to annotate the data set. | | |
| FORTRAN | integer function sfid2ref(sds\_id) | | |
|  | integer sds\_id | | |

SDidtype

hdf\_idtype\_t SDidtype(int32 obj\_id)

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Identifier of the object |
| Purpose | Given an identifier, return the type of object the identifier represents. | | |
| Return value | Returns a value of type hdf\_idtype\_t. | | |
| Description | SDidtype returns a value of type hdf\_idtype\_t, which can be one of the following:  o NOT\_SDAPI\_ID (or -1) - not an SD API identifier  o SD\_ID (or 0) - SD identifier  o SDS\_ID (or 1) - SDS identifier  o DIM\_ID (or 2) - Dimension identifier  SDidtype returns NOT\_SDAPI\_ID for either  + when obj\_id is not a valid HDF identifier, or  + when obj\_id is a valid HDF identifier, but not one of the identifier types in  the SD interface, which are SD identifier, SDS identifier, and  dimension identifier. | | |
| FORTRAN | integer function sfidtype(obj\_id, obj\_type) | | |
|  | integer obj\_id, obj\_type | | |

SDiscoordvar/sfiscvar

intn SDiscoordvar(int32 *sds\_id*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| Purpose | Determines if a data set is a coordinate variable. | | |
| Return value | Returns TRUE (or 1) if the data set is a coordinate variable, and FALSE (or 0) otherwise. | | |
| Description | **SDiscoordvar** determines if the data set specified by the parameter sds\_id is a coordinate variable. | | |
|  | Coordinate variables are created to store metadata associated with dimensions. To ensure compatibility with netCDF, coordinate variables are implemented as data sets. | | |
| FORTRAN | integer function sfiscvar(sds\_id) | | |
|  | integer sds\_id | | |

SDisdimval\_bwcomp/sfisdmvc

intn SDisdimval\_bwcomp(int32 *dim\_id*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| Purpose | Determines whether a dimension has the old and new representations or the new representation only. | | |
|  | Refer to Chapter 3, "Scientific Data Sets (SD API)" of the HDF User’s Guide, for information on old and new dimension representations. | | |
| Return value | Returns SD\_DIMVAL\_BW\_COMP (or 1) if backward compatible, SD\_DIMVAL\_BW\_INCOMP (or 0) if incompatible, FAIL (or -1) if error. | | |
| Description | **SDisdimval\_bwcomp** will flag the dimension specified by the parameter dim\_id as backward-compatible if a vdata with a class name of DIM\_VALS (or “DimVal0.0”) does not exist in the vgroup for that dimension. If the vdata does exist, the specified dimension will be identified by **SDisdimval\_bcomp** as backward-incompatible. | | |
|  | The compatibility mode can be changed by calls to **SDsetdimval\_comp** at any time between the calls to **SDstart** and **SDend**. | | |
| FORTRAN | integer function sfisdmvc(dim\_id) | | |
|  | integer dim\_id | | |

SDisrecord/sfisrcrd

int32 SDisrecord(int32 *sds\_id*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |

|  |  |
| --- | --- |
| Purpose | Determines whether a data set is appendable. |
| Return value | Returns TRUE (or 1) if the data set is appendable, and FALSE (or 0) otherwise. |
| Description | **SDisrecord** will determine if the data set specified by the parameter sds\_id is appendable, which means that the slowest-changing dimension was declared unlimited when the data set was created. |

|  |  |
| --- | --- |
| FORTRAN | integer sfisrcrd(sd\_id) |
|  | integer sd\_id |

SDnametoindex/sfn2index

int32 SDnametoindex(int32 *sd\_id*, char \**sds\_name*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by **SDstart** |
| sds\_name | IN: | Name of the data set |
| Purpose | Determines the index of a data set given its name. | | |
| Return value | Returns the index of the data set (sds\_index) if the data set is found and FAIL (or -1) otherwise. | | |
| Description | SDnametoindex returns the index of the data set with the name specified by the parameter sds\_name. The routine does not accept wildcards in the specified data set name. It also searches on that name in a case-sensitive manner. If there are more than one data set with the same name, the routine will return the index of the first one. | | |
|  | Note that if there are more than one data set with the same name in the file, writing to a data set returned by this function without verifying that it is the desired data set could cause data corruption. | | |
|  | SDgetnumvars\_byname can be used to get the number of data sets (or variables, which includes both data sets and coordinate variables) with the same name. SDnametoindices can be used to get a list of structures containing the indices and the types of all the variables of that same name. | | |
| FORTRAN | integer function sfn2index(sd\_id, sds\_name) | | |
|  | integer sd\_id | | |
|  | character\*(\*) sds\_name | | |

SDnametoindices

intn SDnametoindices(int32 *sd\_id*, char \**sds\_name, varlist\_t \* var\_list*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by **SDstart** |
| sds\_name | IN: | Name of the data set |
| var\_list | OUT: | List of all variables of same name |
| Purpose | Retrieves indices of all variables with the same name. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDnametoindices retrieves a list of structures varlist\_t, containing the indices and the types of all variables of the same name sds\_name.  The structure varlist\_t is defined as:  typedef struct varlist  {  int32 var\_index; /\* index of a variable \*/  vartype\_t var\_type; /\* type of a variable  } varlist\_t;  The type of a variable vartype\_t is defined as:  IS\_SDSVAR (or 0) : variable is an actual SDS  IS\_CRDVAR (or 1) : variable is a coordinate variable  UNKNOWN (or 2) : variable is created before HDF4.2r2, unknown type  The routine does not accept wildcards in the specified data set name. It also searches on that name in a case-sensitive manner. | | |
| FORTRAN | integer function sfn2indices(sd\_id, sds\_name, var\_list, type\_list, n\_vars) | | |
|  | integer sd\_id | | |
|  | character\*(\*) sds\_name | | |
|  | integer var\_list(\*), type\_list(\*) | | |
|  | integer n\_vars | | |

SDreadattr/sfrnatt/sfrcatt

intn SDreadattr(int32 *obj\_id*, int32 *attr\_index*, VOIDP *attr\_buf*)

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Identifier of the object the attribute is attached to |
| attr\_index | IN: | Index of the attribute to be read |
| attr\_buf | OUT: | Buffer for the attribute values |
| Purpose | Reads the values of an attribute. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDreadattr** reads the values of the attribute specified by the parameter attr\_index and stores the values in the buffer attr\_buf. It is assumed that the user has called **SDattrinfo** to retrieve the number of attribute values and allocate sufficient space for the buffer. Note that the routine does not read a subset of attribute values. | | |
|  | The value of obj\_id can be either an SD interface identifier (sd\_id), returned by **SDstart**, a data set identifier (sds\_id), returned by **SDselect**, or a dimension identifier (dim\_id), returned by **SDgetdimid**. | | |
|  | The value of attr\_index is a positive integer and is less than the total number of attributes. The index value can be obtained using the routines **SDnametoindex** and **SDreftoindex**. The total number of attributes for the object can be obtained using the routines **SDgetinfo**, **SDattrinfo**, **SDdiminfo** and **SDfileinfo**. | | |
|  | Note that this routine returns an array of characters, not a standard null-terminated string. If an application is running in an environment where a null-terminated string is expected, the application must add the null character before saving the string or using it further.  Note that this routine has two FORTRAN-77 versions: **sfrnatt** and **sfrcatt**. The **sfrnatt** routine reads numeric attribute data and **sfrcatt** reads character attribute data. | | |
| FORTRAN | integer function sfrnatt(obj\_id, attr\_index, attr\_buffer) | | |
|  | integer obj\_id, attr\_index | | |
|  | <valid numeric data> attr\_buffer(\*) | | |
|  |  | | |
|  | integer function sfrcatt(obj\_id, attr\_index, attr\_buffer) | | |
|  | integer obj\_id, attr\_index | | |
|  | character\*(\*) attr\_buffer | | |

SDreadchunk/sfrchnk/sfrcchnk

intn SDreadchunk(int32 *sds\_id*, int32 \**origin*, VOIDP *datap*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| origin | IN: | Origin of the chunk to be read |
| datap | OUT: | Buffer for the chunk to be read |
| Purpose | Reads a data chunk from a chunked data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDreadchunk** reads the entire chunk of data from the chunked data set identified by the parameter sds\_id, and stores the data in the buffer datap. Reading starts at the location specified by the parameter origin. **SDreadchunk** is used when an entire chunk of data is to be read. **SDreaddata** is used when the read operation is to be done regardless of the chunking scheme used in the data set. | | |
|  | The parameter *origin* specifies the coordinates of the chunk according to the chunk position in the chunked array. Refer to the Chapter 3, "Scientific Data Sets (SD API)" of the HDF User’s Guide, for a description of the organization of chunks in a data set. | | |
|  | **SDreadchunk** will return FAIL (or -1) when an attempt is made to read from a non-chunked data set. | | |
|  | Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**sfrchnk**) and one for character data (**sfrcchnk**). | | |
| FORTRAN | integer sfrchnk(sds\_id, origin, datap) | | |
|  | integer sds\_id, origin(\*) | | |
|  | <valid numeric data type> datap(\*) | | |
|  |  | | |
|  | integer sfrcchnk(sds\_id, origin, datap) | | |
|  | integer sds\_id, origin(\*) | | |
|  | character\*(\*) datap(\*) | | |

SDreaddata/sfrdata/sfrcdata

intn SDreaddata(int32 *sds\_id*, int32 *start*[], int32 *stride*[], int32 *edge*[], VOIDP *buffer*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| start | IN: | Array specifying the starting location from where 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 |
| buffer | OUT: | Buffer to store the data read |
| Purpose | Reads a subsample of data from a data set or coordinate variable. | | |
| Return value | Returns SUCCEED (or 0) if successful or if the data set or coordinate variable contains no data and FAIL (or -1) otherwise. | | |
| Description | **SDreaddata** reads the specified subsample of data from the data set or coordinate variable identified by the parameter sds\_id. The read data is stored in the buffer buffer. The subsample is defined by the parameters start, stride and edge. | | |
|  | The array start specifies the starting position from where the subsample will be read. Valid values of each element in the array start are from 0 to the size of the corresponding dimension of the data set - 1. The dimension sizes are returned by **SDgetinfo**. | | |
|  | The array edge specifies the number of values to read along each data set dimension. | | |
|  | 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 data set will be read. If one of the elements of the array stride is 2, then every other element along the corresponding dimension of the data set will be read, and so on. Specifying stride value of NULL in the C interface or setting all values of the array stride to 1 in either interface specifies the contiguous reading of data. If all values in the array stride are set to 0 or any value causes striding beyond the end of the associate dimension, **SDreaddata** returns FAIL (or -1). No matter what stride value is provided, data is always placed contiguously in the buffer. | | |
|  | When reading data from a “chunked” data set using **SDreaddata**, consideration should be given to the issues presented in the section on chunking in Chapter 3, "Scientific Data Sets (SD API)" and Chapter 14, "HDF Performance Issues" in the HDF User’s Guide. | | |
|  | Note that there are two FORTRAN-77 versions of this routine; **sfrdata** and **sfrcdata**. The **sfrdata** routine reads numeric scientific data and **sfrcdata** reads character scientific data. | | |
| 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 buffer 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. | | |
|  | It is sometimes necessary to determine whether and how a dataset is compressed and whether the software necessary to read that data is available. The compression method used on the dataset can be determined with SDgetcompinfo/sfgcompress and the availability and configuration of the compression software with HCget\_config\_info. Further information is available in the respective entries in this reference manual. | | |
| Note | **Regarding Szip-compressed data:  SDreaddata** can succeed for an Szip-compressed dataset whether the available Szip library is configured either for encoding/decoding or for decoding-only.  If the available Szip configuration is decode-only, HCget\_config\_info will return only COMP\_DECODER\_ENABLED in compression\_config\_info; the returned flags will not include COMP\_ENCODER\_ENABLED. In such a case, the file must have been opened in read-only mode, i.e. with SDstart(filename, DFACC\_RDONLY).  If the Szip available configuration is encode/decode, HCget\_config\_info will return COMP\_ENCODER\_ENABLED|COMP\_DECODER\_ENABLED. In such a case, the file and dataset can be opened in read/write mode.  See the HCget\_config\_info and SDgetcompinfo/sfgcompress entries in this reference manual for further information. | | |
| Note | **Regarding Szip usage and licensing:**  See http://www.hdfgroup.org/doc\_resource/SZIP/ for information regarding the use of Szip in HDF products and Szip licensing. | | |
| FORTRAN | integer function sfrdata(sds\_id, start, stride, edge, buffer) | | |
|  | integer sds\_id, start(\*), stride(\*), edge(\*) | | |
|  | <valid numeric data type> buffer(\*) | | |
|  |  | | |
|  | integer function sfrcdata(sds\_id, start, stride, edge, buffer) | | |
|  | integer sds\_id, start(\*), stride(\*), edge(\*) | | |
|  | character\*(\*) buffer | | |

SDreftoindex/sfref2index

int32 SDreftoindex(int32 *sd\_id*, int32 *sds\_ref*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by **SDstart** |
| sds\_ref | IN: | Reference number of the data set |
| Purpose | Returns the index of a data set given the reference number. | | |
| Return value | Returns the index of the data set (sds\_index) if the data set is found and FAIL (or -1) otherwise. | | |
| Description | SDreftoindex returns the index of a data set identified by its reference number, sds\_ref. | | |
|  | The value of sds\_index returned by SDreftoindex can be passed to SDselect to obtain a data set identifier (sds\_id). | | |
| FORTRAN | integer function sfref2index(sd\_id, sds\_ref) | | |
|  | integer sd\_id, sds\_ref | | |

SDreset\_maxopenfiles

intn SDreset\_maxopenfiles(intn req\_max)

|  |  |  |
| --- | --- | --- |
| req\_max | IN: | Requested maximum number of opened files allowed |
| Purpose | Resets the maximum number of files can be opened at the same time. | | |
| Return value | Returns the current maximum number of opened files allowed if successful and FAIL (or -1) otherwise. | | |
| Description | Prior to release 4.2r2, the maximum number of files that can be opened at the same time was limited to 32. In HDF 4.2r2 and later versions, if this limit is reached, the library will increase it to the system limit minus 3 to account for stdin, stdout, and stderr.  This function can be called anytime to change the maximum number of open files allowed in HDF to req\_max. If req\_max is 0, SDreset\_maxopenfiles will simply return the current maximum number of open files allowed. If req\_max exceeds system limit, SDreset\_maxopenfiles will reset the maximum number of open files to the system limit, and return that value.  Furthermore, if the system maximum limit is reached, the library will push the error code DFE\_TOOMANY onto the error stack. User applications can detect this after an SDstart fails. | | |
| FORTRAN | integer function sfrmaxopenf(req\_max) | | |
|  | integer req\_max | | |

SDselect/sfselect

int32 SDselect(int32 *sd\_id*, int32 *sds\_index*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by **SDstart** |
| sds\_index | IN: | Index of the data set |
| Purpose | Obtains the data set identifier (sds\_id) of a data set. | | |
| Return value | Returns the data set identifier (sds\_id) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDselect** obtains the data set identifier (sds\_id) of the data set specified by its index, sds\_index. | | |
|  | The integration with netCDF has required that a dimension (or coordinate variable) is stored as a data set in the file. Therefore, the value of sds\_index may correspond to the coordinate variable instead of the actual data set. Users should use the routine **SDiscoordvar** to determine whether the given data set is a coordinate variable. | | |
|  | The value of sds\_index is greater than or equal to 0 and less than the number of data sets in the file. The total number of data sets in a file may be obtained from a call to **SDfileinfo**. The **SDnametoindex** routine can be used to find the index of a data set if its name is known. However, when multiple data sets have the same name, SDnametoindices can be used to obtains all the indices. | | |
| FORTRAN | integer function sfselect(sd\_id, sds\_index) | | |
|  | integer sd\_id, sds\_index | | |

SDsetaccesstype/sdfsacct

intn SDsetaccesstype(int32 *sds\_id*, uintn *access\_type*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| accesstype | IN: | Access type |
| Purpose | Sets the I/O access type of an SDS. | | |
| Return value | Returns SUCCEED (or 0) if the SDS data can be accessed via access\_type and FAIL (or -1) otherwise. | | |
| Description | **SDsetaccesstype** sets the type of I/O (serial, paralle,...) for accessing the data of the data set identified by sds\_id. Access types can be DFACC\_SERIAL (or 1), DFACC\_PARALLEL (or 11), and DFACC\_DEFAULT (or 0). | | |
| FORTRAN | integer function sdfsacct(sds\_id, access\_type) | | |
|  | integer sds\_id, access\_type | | |

SDsetattr/sfsnatt/sfscatt

intn SDsetattr(int32 *obj\_id*, char \**attr\_name*, int32 *ntype*, int32 *count*, VOIDP *values*)

|  |  |  |
| --- | --- | --- |
| obj\_id | IN: | Identifier of the object the attribute is to be attached to |
| attr\_name | IN: | Name of the attribute |
| ntype | IN: | Number type of the values in the attribute |
| count | IN: | Total number of values to be stored in the attribute |
| values | IN: | Data values to be stored in the attribute |
| Purpose | Attaches an attribute to an object. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetattr** attaches the attribute to the object specified by the obj\_id parameter. The attribute is defined by its name, attr\_name, number type, ntype, number of attribute values, count, and the attribute values, values. **SDsetattr** provides a generic way for users to define metadata. It implements the label = value data abstraction. | | |
|  | The value of obj\_id can be an SD interface identifier (sd\_id), returned by **SDstart**, a data set identifier (sds\_id), returned by **SDcreate** or **SDselect**, or a dimension identifier (dim\_id), returned by **SDgetdimid**. | | |
|  | If the parameter obj\_id is  - an SD interface identifier (sd\_id,) a global attribute will be created which applies to all objects in the file  - a data set identifier (sds\_id,) an attribute will be attached to the specified data set  - a dimension identifier (dim\_id,) an attribute will be attached to the specified dimension. | | |
|  | The attr\_name argument can be any ASCII string with maximum length of H4\_MAX\_NC\_NAME (or 256). | | |
|  | The ntype parameter can contain any number type supported by the HDF library. These number 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 count parameter. If more than one value is stored, all values must have the same number type. If an attribute with the given name, number type and number of values exists, it will be overwritten. | | |
| Note | Starting in version 4.2.6, SDsetattr will fail immediately when count is 0. In previous releases, SDsetattr did not fail immediately but SDend would fail eventually, which might corrupt the file. | | |
|  | As suggested by a user whose application needed to create an attribute containing character string with zero length, a C application can pass in a single character string containing the '\0' character for values and 1 for count. | | |
|  | Note that there are two FORTRAN-77 versions of this routine; **sfsnatt** and **sfscatt**. The **sfsnatt** routine writes numeric attribute data and **sfscatt** writes character attribute data. | | |
| FORTRAN | integer function sfsnatt(obj\_id, attr\_name, ntype, count, values) | | |
|  | integer obj\_id, ntype, count | | |
|  | character\*(\*) attr\_name | | |
|  | <valid numeric data type> values(\*) | | |
|  |  | | |
|  | integer function sfscatt(obj\_id, attr\_name, ntype, count, values) | | |
|  | integer obj\_id, ntype, count | | |
|  | character\*(\*) attr\_name, values | | |

SDsetblocksize/sfsblsz

intn SDsetblocksize(int32 *sd\_id*, int32 *block\_size*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by **SDstart** |
| block\_size | IN: | Size of the block in bytes |
| Purpose | Sets the block size used for storing data sets with unlimited dimensions. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetblocksize** sets the block size defined in the parameter block\_size for all data sets in the file. **SDsetblocksize** is used when creating new data sets only; it has no effect on pre-existing data sets. | | |
|  | **SDsetblocksize** must be used after calls to **SDcreate** or **SDselect** and before the call to **SDwritedata**. | | |
|  | The block\_size parameter should be set to a multiple of the desired buffer size. | | |
| FORTRAN | integer sfsblsz(sd\_id, block\_size) | | |
|  | integer sd\_id, block\_size | | |

SDsetcal/sfscal

intn SDsetcal(int32 *sds\_id*, float64 *cal*, float64 *cal\_err*, float64 *offset*, float64 *offset\_err*, int32 *ntype*)

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| sds\_id | IN: | | Data set identifier returned by **SDcreate** or **SDselect** | | |
| cal | IN: | | Calibration factor | | |
| cal\_err | IN: | | Calibration error | | |
| offset | IN: | | Uncalibrated offset | | |
| offset\_err | IN: | | Uncalibrated offset error | | |
| ntype | IN: | | Number type of uncalibrated data | | |
| Purpose | Sets the calibration information. | | | | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | | | | |
| Description | **SDsetcal** stores the calibration record associated with a data set. A calibration record contains the following information: | | | | | |
| cal | | Calibration factor |
| cal\_err | | Calibration error |
| offset | | Uncalibrated offset |
| offset\_err | | Uncalibrated offset error |
| ntype | | Number type of uncalibrated data |
|  | The relationship between a value cal\_value stored in a data set and the original value is defined as: orig\_value = cal \* (cal\_value - offset). | | | | | |
|  | The variable offset\_err contains a potential error of offset, and cal\_err contains a potential error of cal. Currently the calibration record is provided for information only. The SD interface performs no operations on the data based on the calibration tag. | | | | | |
|  | The calibration information is automatically cleared after a call to **SDreaddata** or **SDwritedata**. Therefore, **SDsetcal** must be called once for each data set that is to be read or written. | | | | | |
| FORTRAN | integer function sfscal(sds\_id, cal, cal\_err, offset, offset\_err, ntype) | | | | | |
|  | integer sds\_id, ntype | | | | | |
|  | real\*8 cal, cal\_err, offset, offset\_err | | | | | |

SDsetchunk/sfschnk

intn SDsetchunk(int32 *sds\_id*, HDF\_CHUNK\_DEF *cdef*, int32 *flag*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
|  |  |  |
| **C only:** |  |  |
| cdef | IN: | Pointer to the chunk definition |
| flag | IN: | Compression flag |
|  |  |  |
| **Fortran only:** |  |  |
| dim\_length | IN: | Chunk dimensions array |
| comp\_type | IN: | Type of compression |
| comp\_prm | IN: | Compression parameters array |
| Purpose | Sets the chunk size and the compression method, if any, of a data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetchunk** makes the data set specified by the parameter sds\_id a chunked data set according to the chunking and compression information provided in the parameters cdef and flag in C, and in the parameters comp\_type and comp\_prm in Fortran. | | |
|  |  | | |
|  | C only: | | |
|  | The parameter flag specifies the type of the data set, i.e., if the data set is chunked or chunked and compressed with either RLE, Skipping Huffman, GZIP, Szip, or NBIT compression methods. Valid values of flag are as follows:  o HDF\_CHUNK for a chunked data set with no compression  o HDF\_CHUNK | HDF\_COMP for a chunked data set compressed with RLE, Skipping Huffman, GZIP, or Szip compression methods  o HDF\_CHUNK | HDF\_NBIT for a chunked and NBIT-compressed data set | | |
|  | Chunking and compression information are passed in the parameter cdef. The parameter cdef has a type of HDF\_CHUNK\_DEF, defined in the HDF library 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 | | |
|  | There are three pieces of chunking and compression information which should be specified: chunking dimensions, compression type, and, if needed, compression parameters. | | |
|  | If the data set is chunked, i.e., flag value is HDF\_CHUNK, then chunk\_lengths[] elements of cdef union (cdef.chunk\_lengths[]) have to be initialized to the chunk dimensions. | | |
|  | If the data set is chunked and compressed using RLE, Skipping Huffman, Szip, or GZIP methods (i.e., flag value is set up to HDF\_CHUNK | HDF\_COMP), then the elements chunk\_lengths[] of the structure comp in the union cdef (cdef.comp.chunk\_lengths[]) have to be initialized to the chunk dimensions. | | |
|  | If the data set is chunked and NBIT compression is applied (i.e., flag values is set up to HDF\_CHUNK | HDF\_NBIT), then the elements chunk\_lengths[] of the structure nbit in the union cdef (cdef.nbit.chunk\_lengths[]) have to be initialized to the chunk dimensions. | | |
|  | Compression types are passed in the field comp\_type of the structure cinfo , which is an element of the structure comp in the union cdef (cdef.comp.cinfo.comp\_type). Refer to the **SDsetcompress** page in this manual for the definition of structure comp\_info. Valid compression methods are:  COMP\_CODE\_NONE for no compression  COMP\_CODE\_RLE for RLE run-length encoding  COMP\_CODE\_SKPHUFF for Skipping Huffman compression  COMP\_CODE\_DEFLATE for GZIP compression  COMP\_CODE\_SZIP 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. | | |
|  | NBIT compression parameters are specified in the fields start\_bit, bit\_len, sign\_ext, and fill\_one in the structure nbit of the union cdef. | | |
|  | Fortran only: | | |
|  | The dim\_length array specifies the chunk dimensions. | | |
|  | The comp\_type parameter specifies the compression type. Valid compression types and their values are defined in the hdf.inc file, and are listed below: | | |
|  | COMP\_CODE\_NONE (or 0) for no compression  COMP\_CODE\_RLE (or 1) for RLE compression algorithm  COMP\_CODE\_NBIT (or 2) for NBIT compression algorithm  COMP\_CODE\_SKPHUFF (or 3) for Skipping Huffman compression  COMP\_CODE\_DEFLATE (or 4) for GZIP compression algorithm  COMP\_CODE\_SZIP (or 5) for Szip compression algorithm | | |
|  | The comp\_prm(1) parameter specifies the skipping size for the Skipping Huffman compression method and the deflate level for the GZIP compression method. The skipping size value must be 1 or greater; the deflate level must be an integer value between 0 and 9 inclusive. | | |
|  | For NBIT compression, the four elements of the array comp\_prm correspond to the four NBIT compression parameters listed in the structure nbit. The value of comp\_prm(1) should be set to the value of start\_bit, the value of comp\_prm(2) should be set to the value of bit\_len, the value of comp\_prm(3) should be set to the value of sign\_ext, and the value of comp\_prm(4) should be set to the value of fill\_one. See the HDF\_CHUNK\_DEF union description and the description of **SDsetnbitdataset** function for NBIT compression parameters definitions. | | |
|  | For Szip compression, the first two elements of the array comp\_prm correspond to the first two Szip compression parameters listed in the structure szip. The value of comp\_prm(1) should be set to the value of option\_mask and the value of comp\_prm(2) should be set to the value of pixels\_per\_block. | | |
| FORTRAN | integer sfschnk(sds\_id, dim\_length, comp\_type, comp\_prm) | | |
|  | integer sds\_id, dim\_length, comp\_type, comp\_prm(\*) | | |

SDsetchunkcache/sfscchnk

intn SDsetchunkcache(int32 *sds\_id*, int32 *maxcache*, int32 *flag*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| maxcache | IN: | Maximum number of chunks in the cache |
| flag | IN: | Flag determining the behavior of the routine |
| Purpose | Sets the size of the chunk cache. | | |
| Return value | Returns the maximum number of chunks that can be cached (the value of the parameter maxcache) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetchunkcache** sets the size of the chunk cache to the value of the parameter maxcache. | | |
|  | Currently the only allowed value of the parameter flag is 0, which designates default operation. | | |
|  | By default, when a generic data set is promoted to be a chunked data set, the parameter maxcache is set to the number of chunks along the fastest changing dimension and a cache for the chunks is created. | | |
|  | If the chunk cache is full and the value of the parameter maxcache is greater then the current maxcache value, then the chunk cache is reset to the new value of maxcache. Otherwise the chunk cache remains at the current value of maxcache. If the chunk cache is not full, then the chunk cache is set to the new value of maxcache only if the new maxcache value is greater than the current number of chunks in the cache. | | |
|  | Do not set the value of maxcache to be less than the number of chunks along the fastest-changing dimension of the biggest slab to be written or read via **SDreaddata** or **SDwritedata**. Doing this will cause internal thrashing. See the section on chunking in Chapter 14, "HDF Performance Issues" in the HDF User’s Guide, for more information on this. | | |
| FORTRAN | integer sfscchnk(sds\_id, maxcache, flag) | | |
|  | integer sds\_id, maxcache, flag | | |

SDsetcompress/sfscompress

intn SDsetcompress(int32 *sds\_id*, int32 *comp\_type*, comp\_info \**c\_info*)

|  |  |  |  |
| --- | --- | --- | --- |
| sds\_id | | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| comp\_type | | IN: | Compression method |
|  | |  |  |
| *C only:* | |  |  |
| c\_info | | IN: | Pointer to the comp\_info union |
|  | |  |  |
| *Fortran only:* | |  |  |
| comp\_prm | | IN: | Compression parameters array |
| Purpose | Compresses the data set with the specified compression method. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetcompress** compresses the data set identified by the parameter sds\_id according to the compression method specified by the parameter comp\_type and the compression information specified by the parameter c\_info in C and comp\_prm in Fortran. **SDsetcompress** sets up the special element for the compressed data written during the next call to **SDwritedata**. | | |
|  | **SDsetcompress** is a simplified interface to the **HCcreate** routine and should be used instead of **HCcreate**, unless the user is familiar with working with the lower-level routines. | | |
|  | The parameter comp\_type is the compression type definition and is set to one of the following:  COMP\_CODE\_RLE (or 1) --for run-length encoding (RLE)  COMP\_CODE\_SKPHUFF (or 3) --for Skipping Huffman  COMP\_CODE\_DEFLATE (or 4) --for GZIP compression  COMP\_CODE\_SZIP (or 5) --for Szip compression | | |
|  | 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 SDsetcompress \*/} jpeg;  struct  {/\* Not used by SDsetcompress \*/} 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; | | |
|  | The skipping size for the Skipping Huffman algorithm must be 1 or greater and 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 c\_info.deflate.level field in C and in the parameter comp\_prm(1) in Fortran. Valid values are integers between 0 and 9 inclusive. | | |
|  | The Szip options mask and the number of pixels per block in a chunked and Szip-compressed dataset are specified in c\_info.szip.options\_mask and c\_info.szip.pixels\_per\_block, respectively. | | |
|  | The options mask can contain either of the following values:  SZ\_EC\_OPTION\_MASK - Specifies entropy coding method  SZ\_NN\_OPTION\_MASK - Specifies nearest neighbor coding method | | |
|  | The following guidelines may be helpful in selecting the encoding method:  o The entropy coding method, the EC option specified by SZ\_EC\_OPTION\_MASK, is best suited for data that has been processed. The EC method works best for small numbers.  o The nearest neighbor coding method, the NN option specified by SZ\_NN\_OPTION\_MASK, preprocesses the data then applies the EC method as above.  Other factors may affect results, but the above criteria provide a good starting point for optimizing data compression. | | |
|  | The Szip values of the number of pixels per scanline, the number of bits in a pixel, and the number of pixels in an image, are computed by the HDF4 library and provided to the user in c\_info.szip.pixels\_per\_scanline, c\_info.szip.bits\_per\_pixel, and c\_info.szip.pixels, respectively. | | |
|  | **SDsetcompress** will succeed in setting Szip compression for a dataset only if the Szip library is available and configured for encoding, i.e., **HCget\_config\_info** must return the flag COMP\_DECODER\_ENABLED|COMP\_ENCODER\_ENABLED in compression\_config\_info. | | |
|  | Compression is not supported for unlimited dimension SDSs. SDsetcompress will fail on an SDS with unlimited dimension. If the application proceeds after such call, subsequent SDwritedata will write uncompressed data to the SDS. | | |
| Note | **Regarding Szip usage and licensing:**  See http://www.hdfgroup.org/doc\_resource/SZIP/ for information regarding the use of Szip in HDF products and Szip licensing. | | |
| FORTRAN | integer sfscompress(sds\_id, comp\_type, comp\_prm) | | |
|  | integer sds\_id, comp\_type, comp\_prm(\*) | | |

SDsetdatastrs/sfsdtstr

intn SDsetdatastrs(int32 *sds\_id*, char \**label*, char \**unit*, char \**format*, char \**coordsys*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| label | IN: | Label (predefined attribute) |
| unit | IN: | Unit (predefined attribute) |
| format | IN: | Format (predefined attribute) |
| coordsys | IN: | Coordinate system (predefined attribute) |
| Purpose | Sets the predefined attributes for a data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetdatastrs** sets the predefined attributes of the data set, identified by sds\_id, to the values specified in the parameters label, unit, format and coordsys. The predefined attributes are label, unit, format, and coordinate system. If the user does not want a string returned, the corresponding parameter can be set to NULL in C and an empty string in Fortran. | | |
|  | For more information about predefined attributes, refer to Section 3.10, "Predefined Attributes" of the HDF User’s Guide. | | |
| FORTRAN | integer function sfsdtstr(sds\_id, label, unit, format, coordsys) | | |
|  | integer sds\_id | | |
|  | character\*(\*) label, unit, format, coordsys | | |

SDsetdimname/sfsdmname

intn SDsetdimname(int32 *dim\_id*, char \**dim\_name*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| dim\_name | IN: | Name of the dimension |
| Purpose | Assigns a name to a dimension. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetdimname** sets the name of the dimension identified by the parameter dim\_id to the value specified in the parameter dim\_name. Dimensions that are not explicitly named by the user will have the default name of “fakeDim[x]” specified by the HDF library, where [x] denotes the dimension index. | | |
|  | If another dimension exists with the same name it is assumed that they refer to the same dimension object and changes to one will be reflected in the other. If the dimension with the same name has a different size, an error condition will result. | | |
|  | The length of the parameter dim\_name can be at most 64 characters. | | |
|  | Naming dimensions is optional but encouraged. | | |
| Note | **Regarding naming a dimension the same as an SDS’ name:**  Prior to HDF4.2r2, when a dimension was named the same as that of a one-dimensional SDS, data corruption will occur after certain operations, such as setting attribute or setting dimension scale. The corrupted data was unrecoverable. However, this problem has been fixed for future data. | | |
| FORTRAN | integer function sfsdmname(dim\_id, dim\_name) | | |
|  | integer dim\_id | | |
|  | character\*(\*) dim\_name | | |

SDsetdimscale/sfsdscale

intn SDsetdimscale(int32 *dim\_id*, int32 *count*, int32 *ntype*, VOIDP *data*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| count | IN: | Total number of values along the dimension |
| ntype | IN: | Number type of the values along the dimension |
| data | IN: | Value of each increment along the dimension |
| Purpose | Stores the values of a dimension. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | SDsetdimscale stores scale information for the dimension identified by the parameter dim\_id. Note that it is possible to store dimension scale values without naming the dimension. | | |
|  | For fixed-size arrays, the value of count must be equal to the the dimension size or the routine will fail. | | |
|  | Note that, due to the existence of the parameter ntype, the dimension scales need not have the same type as the data set. | | |
|  | Note that if SDsetdimscale is called and SDsetdimname is subsequently called for the same dimension, SDsetdimscale must be called again to reassociate the scale with the new name. | | |
| FORTRAN | integer function sfsdscale(dim\_id, count, ntype, data) | | |
|  | integer dim\_id, count, ntype | | |
|  | <valid data type> data(\*) | | |

SDsetdimstrs/sfsdmstr

intn SDsetdimstrs(int32 *dim\_id*, char \**label*, char \**unit*, char \**format*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| label | IN: | Label (predefined attribute) |
| unit | IN: | Unit (predefined attribute) |
| format | IN: | Format (predefined attribute) |
| Purpose | Sets the predefined attribute of a dimension. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetdimstrs** sets the predefined attribute (label, unit, and format) for a dimension and its scale to the values specified in the parameters label, unit and format. If a parameter is set to NULL in C and an empty string in Fortran, then the attribute corresponding to that parameter will not be written. For more information about predefined attributes, refer to Section 3.10, "Predefined Attributes" of the HDF User’s Guide. | | |
| FORTRAN | integer function sfsdmstr(dim\_id, label, unit, format) | | |
|  | integer dim\_id | | |
|  | character\*(\*) label, unit, format | | |

SDsetdimval\_comp/sfsdmvc

intn SDsetdimval\_comp(int32 *dim\_id*, intn *comp\_mode*)

|  |  |  |
| --- | --- | --- |
| dim\_id | IN: | Dimension identifier returned by **SDgetdimid** |
| comp\_mode | IN: | Compatibility mode to be set |
| Purpose | Determines whether a dimension *will have* the old and new representations or the new representation only. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetdimval\_comp** sets the compatibility mode specified by the comp\_mode parameter for the dimension identified by the dim\_id parameter. The two possible compatibility modes are: “backward-compatible” mode, which implies that the old and new dimension representations are written to the file, and “backward-incompatible” mode, which implies that only the new dimension representation is written to the file. | | |
|  | Unlimited dimensions are always backward-compatible, therefore **SDsetdimval\_comp** takes no action on unlimited dimensions. | | |
|  | As of HDF version 4.1r1, the default mode is backward-incompatible. Subsequent calls to **SDsetdimval\_comp** will override the settings established in previous calls to the routine. | | |
|  | The comp\_mode parameter can be set to SD\_DIMVAL\_BW\_COMP (or 1), which specifies backward-compatible mode, or SD\_DIMVAL\_BW\_INCOMP (or 0), which specifies backward-incompatible mode. | | |
| FORTRAN | integer function sfsdmvc(dim\_id, comp\_mode) | | |
|  | integer dim\_id, comp\_mode | | |

SDsetexternalfile/sfsextf

intn SDsetexternalfile(int32 *sds\_id*, char \**filename*, int32 *offset*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| filename | IN: | Name of the external file |
| offset | IN: | Number of bytes from the beginning of the external file to where the data will be written |
| Purpose | Stores data in an external file. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetexternalfile** allows users to move the actual data values (i.e., not metadata) of a data set, sds\_id, into the external data file named by the parameter filename, and started at the offset specified by the parameter offset. The metadata remains in the original file. Note that this routine works only with HDF post-version 3.2 files. | | |
|  | Data can only be moved once for any given data set, and it is the user's responsibility to make sure the external data file is kept with the “original” file. | | |
|  | If the data set 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 data set does not exist, its data will be written to the external file during the consequent calls to **SDwritedata**. | | |
|  | See the reference manual entries for **HXsetcreatedir** and **HXsetdir** for more information on the options available for accessing external files. | | |
| FORTRAN | integer function sfsextf(sds\_id, file\_name, offset) | | |
|  | integer sds\_id, offset | | |
|  | character\*(\*) file\_name | | |

SDsetfillmode/sfsflmd

intn SDsetfillmode(int32 *sd\_id*, intn *fill\_mode*)

|  |  |  |
| --- | --- | --- |
| sd\_id | IN: | SD interface identifier returned by **SDstart** |
| fill\_mode | IN: | Fill mode |
| Purpose | Sets the current fill mode of a file. | | |
| Return value | Returns the fill mode value before it was reset if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetfillmode** applies the fill mode specified by the parameter fill\_mode to all data sets contained in the file identified by the parameter sd\_id. | | |
|  | Possible values of fill\_mode are SD\_FILL (or 0) and SD\_NOFILL (or 256). SD\_FILL is the default mode, and indicates that fill values will be written when the data set is created. SD\_NOFILL indicates that fill values will not be written. | | |
|  | When a data set without unlimited dimensions is created, by default the first **SDwritedata** call will fill the entire data set with the default or user-defined fill value (set by **SDsetfillvalue**). In data sets with an unlimited dimension , if a new write operation takes place along the unlimited dimension beyond the last location of the previous write operation, the array locations between these written areas will be initialized to the user-defined fill value, or the default fill value if a user-defined fill value has not been specified. | | |
|  | If it is certain that all data set values will be written before any read operation takes place, there is no need to write the fill values. Simply call **SDsetfillmode** with fill\_mode value set to SD\_NOFILL, which will eliminate all fill value write operations to the data set. For large data sets, this can improve the speed by almost 50%. | | |
| FORTRAN | integer function sfsflmd(sd\_id, fill\_mode) | | |
|  | integer sd\_id, fill\_mode | | |

SDsetfillvalue/sfsfill/sfscfill

intn SDsetfillvalue(int32 *sds\_id*, VOIDP *fill\_value*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| fill\_value | IN: | Fill value |
| Purpose | Sets the fill value for a data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDsetfillvalue** sets the fill value specified by the fill\_value parameter for the data set identified by the sds\_id parameter. | | |
|  | The fill value is assumed to have the same number type as the data set. | | |
|  | The following are the default fill values for different number types:  FILL\_BYTE (char)-127 /\* Largest Negative value \*/  FILL\_CHAR (char)0  FILL\_SHORT (short)-32767  FILL\_LONG (long)-2147483647  FILL\_FLOAT 9.9692099683868690e+36 /\* near 15 \* 2^119 \*/  FILL\_DOUBLE 9.9692099683868690e+36 | | |
|  | It is recommended to call **SDsetfillvalue** before writing data. | | |
| FORTRAN | integer function sfsfill(sds\_id, fill\_value) | | |
|  | integer sds\_id | | |
|  | <valid numeric data type> fill\_value | | |
|  |  | | |
|  | integer function sfscfill(sds\_id, fill\_value) | | |
|  | integer sds\_id | | |
|  | character\*(\*) fill\_value | | |

SDsetnbitdataset/sfsnbit

intn SDsetnbitdataset(int32 *sds\_id*, intn *start\_bit*, intn *bit\_len*, intn *sign\_ext*, intn *fill\_one*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| start\_bit | IN: | Leftmost bit of the field to be written |
| bit\_len | IN: | Length of the bit field to be written |
| sign\_ext | IN: | Sign extend specifier |
| fill\_one | IN: | Background bit specifier |
| Purpose | Specifies a non-standard bit length for the data set values. | | |
| Return value | Returns a positive value if successful and FAIL (or -1) otherwise. | | |
| Description | SDsetnbitdataset allows the HDF user to specify that the data set identified by the parameter sds\_id contains data of a non-standard length defined by the parameters start\_bit and bit\_len. Additional information about the non-standard bit length decoding are specified in the parameters sign\_ext and fill\_one. | | |
|  | Any length between 1 and 32 bits can be specified. After SDsetnbitdataset has been called for the data set array, any read or write operations will involve a conversion between the new data length of the data set array and the data length of the read or write buffer. | | |
|  | Bit lengths of all number types are counted from the right of the bit field starting with 0. In a bit field containing the values 01111011, bits 2 and 7 are set to 0 and all the other bits are set to 1. | | |
|  | The start\_bitparameter specifies the leftmost position of the variable-length bit field to be written. For example, in the bit field described in the preceding paragraph a start\_bit parameter set to 4 would correspond to the fourth bit value of 1 from the right. | | |
|  | The bit\_len parameter specifies the number of bits of the variable-length bit field to be written. This number includes the starting bit and the count proceeds toward the right end of the bit field - toward the lower-bit numbers. For example, starting at bit 5 and writing 4 bits of the bit field described in the preceding paragraph would result in the bit field 1110 being written to the data set. This would correspond to a start\_bit value of 5 and a bit\_len value of 4. | | |
|  | The sign\_ext parameter specifies whether to use the leftmost bit of the variable-length bit field to sign-extend to the leftmost bit of the data set data. For example, if 9-bit signed integer data is extracted from bits 17-25 and the bit in position 25 is 1, then when the data is read back from disk, bits 26-31 will be set to 1. Otherwise bit 25 will be 0 and bits 26-31 will be set to 0. The sign\_ext parameter can be set to TRUE (or 1) or FALSE (or 0) - specify TRUE to sign-extend. | | |
|  | The fill\_one specifies whether to fill the “background” bits with the value 1 or 0. This parameter can also be set to TRUE or FALSE. | | |
|  | The “background” bits of a variable-length data set are the bits that fall outside of the variable-length bit field stored on disk. For example, if five bits of an unsigned 16-bit integer data set located in bits 5 to 9 are written to disk with the fill\_one parameter set to TRUE (or 1), then when the data is reread into memory bits 0 to 4 and 10 to 15 would be set to 1. If the same 5-bit data was written with a fill\_one value of FALSE (or 0), then bits 0 to 4 and 10 to 15 would be set to 0. | | |
|  | This bit operation is performed before the sign-extend bit-filling. For example, using the sign\_ext example above, bits 0 to 16 and 26 to 31 will first be set to the “background” bit value, and then bits 26 to 31 will be set to 1 or 0 based on the value of the 25th bit. | | |
| FORTRAN | integer function sfsnbit(sds\_id, start\_bit, bit\_len, sign\_ext, fill\_one) | | |
|  | integer sds\_id, start\_bit, bit\_len, sign\_ext, fill\_one | | |

SDsetrange/sfsrange

intn SDsetrange(int32 *sds\_id*, VOIDP *max*, VOIDP *min*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| max | IN: | Maximum value of the range |
| min | IN: | Minimum value of the range |

|  |  |
| --- | --- |
| Purpose | Sets the maximum and minimum range values for a data set. |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. |
| Description | **SDsetrange** sets the maximum and minimum range values of the data set identified by the parameter sds\_id with the values of the parameters max and min. The term “range” is used here to describe the range of numeric values stored in a data set. |
|  | It is assumed that the number type for the maximum and minimum range values are the same as the type of the data. |
|  | This routine does not compute the maximum and minimum range values, it only stores the values as given. As a result, the maximum and minimum range values may not always reflect the actual maximum and minimum range values in the data set data. |

|  |  |
| --- | --- |
| FORTRAN | integer function sfsrange(sds\_id, max, min) |
|  | integer sds\_id |
|  | <valid numeric data type> max, min |

SDstart/sfstart

int32 SDstart(char \**filename*, int32 *access\_mode*)

|  |  |  |
| --- | --- | --- |
| filename | IN: | Name of the HDF file |
| access\_mode | IN: | The file access mode in effect during the current session |
| Purpose | Opens an HDF file and initializes an SD interface. | | |
| Return value | Returns an SD interface identifier if successful and FAIL (or -1) otherwise. | | |
| Description | **SDstart** opens the file with the name specified by the parameter filename, with the access mode specified by the parameter access\_mode, and returns an SD interface identifier (sd\_id). This routine must be called for each file before any other SD calls can be made on that file. | | |
|  | The type of identifier returned by **SDstart** is currently not the same as the identifier returned by **Hopen**. As a result, the SD interface identifiers (sd\_id) returned by this routine are not understood by other HDF interfaces. | | |
|  | To mix SD API calls and other HDF API calls, use **SDstart** and **Hopen** on the same file. **SDstart** must precede all SD calls, and **Hopen** must precede all other HDF function calls. To terminate access to the file, use **SDend** to dispose of the SD interface identifier, sd\_id, and **Hclose** to dispose of the file identifier, file\_id. | | |
|  | The file identified by the parameter filename can be any one of the following: an XDR-based netCDF file, “old-style” DFSD file or a “new-style” SD file. | | |
|  | The value of the parameter access\_mode can be one of the following: | | |
|  | DFACC\_READ - Open existing file for read-only access. If the file does not exist, specifying this mode will cause **SDstart** to return FAIL (or -1).  DFACC\_WRITE - Open existing file for read and write access. If the file does not exist, specifying this mode will cause **SDstart** to return FAIL (or -1).  DFACC\_CREATE - Create a new file with read and write access. If the file has already existed, its contents will be replaced. | | |
| Note | Starting from HDF 4.2r2, the maximum number of open files is no longer limited to 32. It can be up to what the system allowed. | | |
| Note | It has been reported that opening/closing file in loops is very slow; thus, it is not recommended to perform such operations too many times, particularly, when data is being added to the file between opening/closing. | | |
| FORTRAN | integer function sfstart(filename, access\_mode) | | |
|  | character\*(\*) filename | | |
|  | integer access\_mode | | |

SDwritechunk/sfwchnk/sfwcchnk

intn SDwritechunk(int32 *sds\_id*, int32 \**origin*, VOIDP *datap*)

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| origin | IN: | Origin of the chunk to be written |
| datap | IN: | Buffer for the chunk data to be written |
| Purpose | Writes a data chunk to a chunked data set. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDwritechunk** writes the entire chunk of data stored in the buffer datap to the chunked data set identified by the parameter sds\_id. Writing starts at the location specified by the parameter origin. **SDwritechunk** is used when an entire chunk of data is to be written. **SDwritedata** is used when the write operation is to be done regardless of the chunking scheme used in the data set. | | |
|  | **SDwritechunk** will return FAIL (or -1) when an attempt is made to use it to write to a non-chunked data set. | | |
|  | The parameter origin specifies the coordinates of the chunk according to the chunk position in the overall chunk array. Refer to Chapter 3, "Scientific Data Sets (SD API)" in the HDF User’s Guide, for a description of the organization of chunks in a data set. | | |
|  | Note that there are two FORTRAN-77 versions of this routine; one for numeric data (**sfwchnk**) and one for character data (**sfwcchnk**). | | |
| Note | **Regarding Szip-compressed data:  SDwritechunk** can succeed only when the available Szip library is configured for encoding/decoding, i.e., when HCget\_config\_info returns COMP\_ENCODER\_ENABLED|COMP\_DECODER\_ENABLED in compression\_config\_info.  See the SDgetcompinfo/sfgcompress and HCget\_config\_info entries in this reference manual for further discussion of compression methods and configuration. | | |
| Note | **Regarding Szip usage and licensing:**  See http://www.hdfgroup.org/doc\_resource/SZIP/ for information regarding the use of Szip in HDF products and Szip licensing. | | |
| FORTRAN | integer sfwchnk(sds\_id, origin, datap) | | |
|  | integer sds\_id, origin | | |
|  | <valid numeric data type> datap(\*) | | |
|  |  | | |
|  | integer sfwcchnk(sds\_id, origin, datap) | | |
|  | integer sds\_id, origin | | |
|  | character\*(\*) datap(\*) | | |

SDwritedata/sfwdata/sfwcdata

intn SDwritedata(int32 *sds\_id*, int32 *start*[], int32 *stride*[], int32 *edge*[], VOIDP *buffer* )

|  |  |  |
| --- | --- | --- |
| sds\_id | IN: | Data set identifier returned by **SDcreate** or **SDselect** |
| start | IN: | Array specifying the starting location of the data to be written |
| stride | IN: | Array specifying the number of values to skip along each dimension |
| edge | IN: | Array specifying the number of values to be written along each dimension |
| buffer | IN: | Buffer for the values to be written |
| Purpose | Writes a subsample of data to a data set or to a coordinate variable. | | |
| Return value | Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise. | | |
| Description | **SDwritedata** writes the specified subsample of data to the data set or coordinate variable identified by the parameter sds\_id. The data is written from the buffer buffer. The subsample is defined by the parameters start, stride and edge. | | |
|  | The array start specifies the starting position from where the subsample will be written. Valid values of each element in the array start are from 0 to the size of the corresponding dimension of the data set - 1. The dimension sizes are returned by **SDgetinfo**. | | |
|  | The array edge specifies the number of values to write along each data set dimension. | | |
|  | 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 data set will be written. If one of the elements of the array stride is 2, then every other element along the corresponding dimension of the data set will be written, and so on. Specifying stride value of NULL in the C interface or setting all values of the array stride to 1 in either interface specifies the contiguous writing of data. If all values in the array stride are set to 0, **SDwritedata** returns FAIL (or -1). | | |
|  | When writing data to a chunked data set using **SDwritedata**, consideration should be given to the issues presented in the section on chunking in Chapter 3, "Scientific Data Sets (SD API)" and Chapter 14, "HDF Performance Issues" in the HDF User’s Guide. | | |
|  | Note that there are two FORTRAN-77 versions of this routine; **sfwdata** and **sfwcdata**. The **sfwdata** routine writes numeric data and **sfwcdata** writes character scientific data. | | |
| 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. | | |
| Note | Regarding compressed data sets:  If a data set is compressed, it may be necessary to determine whether the compression method is available on the current system and configured so that data can be encoded before being written. The compression method can be determined through the use of SDgetcompinfo and the configuration of that method on the current system through HCget\_config\_info.  Partial writing is not allowed on compressed data set. To partially modify data, an application can read the data set, modify the specific values in the buffer, then re-write the entire data set. | | |
| Note | Regarding Szip-compressed data:  SDwritedata can succeed only when the available Szip library is configured for encoding/decoding, i.e., when HCget\_config\_info returns COMP\_ENCODER\_ENABLED|COMP\_DECODER\_ENABLED in compression\_config\_info. | | |
| Note | Regarding Szip usage and licensing:  See http://www.hdfgroup.org/doc\_resource/SZIP/ for information regarding the use of Szip in HDF products and Szip licensing. | | |
| FORTRAN | integer function sfwdata(sds\_id, start, stride, edge, buffer) | | |
|  | integer sds\_id | | |
|  | integer start(\*), stride(\*), edge(\*) | | |
|  | <valid numeric data type> buffer(\*) | | |
|  |  | | |
|  | integer function sfwcdata(sds\_id, start, stride, edge, buffer) | | |
|  | integer sds\_id | | |
|  | integer start(\*), stride(\*), edge(\*) | | |
|  | character\*(\*) buffer(\*) | | |