sfgainfo

SDattrinfo

```
intn SDattrinfo(int32 s_id, int32 attr_index, char *attr_name, int32 *number_type,
          int32 *count)
```

s_id	IN:	An *sd_id* for a global attribute; *sds_id* for a local attribute
attr_index	IN:	Index of the attribute to read
attr_name	OUT:	Space to hold the name assigned to the attribute/data set/dimension
number_type	OUT:	Space to hold the number type for the attribute values
count	OUT:	Space to hold the total number of values in the specified attribute

Purpose — Gets information about a global or local attribute.

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

Description — This routine should be used before reading the value of an attribute with **`SDreadattr`**. **`SDreadattr`** requires a buffer to hold the values of the attribute. The size of this buffer should be at least `count*DFKNTsize(number_type)` bytes long.

FORTRAN

```
integer function sfgainfo(s_id, attr_index, attr_name, number_type,
count)
    character*(*)  attr_name
    integer*4      s_id, attr_index, number_type, count
```

SDcreate

sfcreate

SDcreate

```
int32 SDcreate(int32 sd_id, char *name, int32 number_type, int32 rank,
          int32 dimsizes[])
```

sd_id	IN:	The SD interface ID returned from **SDstart**
name	IN:	ASCII string defining a variable name
number_type	IN:	Number type for the values in the data set
rank	IN:	The number of dimensions in the data set
dimsizes	IN:	The size of each dimension

Purpose — Creates a new data set.

Return value — Returns the *sds_id* if successful and FAIL(-1) otherwise.

Description — If name is NULL a fake name will be generated. The name will be truncated to the MAX_NC_NAME length as specified in the HDF header files.

Once a data set has been created, it is impossible 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. If you wish to add data or modify an existing data set, use **SDselect** to get the *sds_id*.

In C, if *dimsizes*[0] is assigned the value SD_UNLIMITED then this dimension is considered to be an "unlimited dimension" and the user can append data along this dimension at will. In Fortran, *dimsizes*(*rank*) is the only dimension that can be unlimited. This is useful when the eventual size of a data set is not known at creation time.

MAX_VAR_DIMS is the maximum rank a data set can have.

FORTRAN

```
integer function sfcreate(sd_id, name, number_type, rank, dimsizes)
    character*(*)  name
    integer*4      sd_id, number_type, rank, dimsizes(*)
```

sfgdinfo

SDdiminfo

```
intn SDdiminfo(int32 dim_id, char *name, int32 *count, int32 *number_type,
          int32 *nattrs)
```

dim_id	IN:	Dimension ID returned from **SDgetdimid**
name	OUT:	Array to retrieve dimension name
count	OUT:	Space allocated to store the size of the dimension
number_type	OUT:	Space allocated to store the number type for the data stored in the data set
nattrs	OUT:	Space for attribute count assigned to the dimension's coordinate variable

Purpose — Gets information about a dimension.

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

Description — If scale information has been stored for this dimension via **SDsetdimscale**, *number_type* will contain the number type otherwise the *number_type* will be `0`. If the user has not named the dimension via **SDsetdimname** a place holder will be generated. If the name is unimportant, `NULL` can be passed in for the *name* parameter.

FORTRAN

```
integer function sfgdinfo(dim_id, name, count, number_type,
nattrs)
    character*(*)  name
    integer*4      dim_id, count, number_type, nattrs
```

SDend

sfend

SDend

```
intn SDend(int32 sd_id)
```

sd_id	IN:	The SD interface ID returned from **SDstart**

Purpose — Closes the file and cleans up memory.

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

Description — **SDend** closes the file when done with SD interface activities. If a user program exits without calling this function, changes made to the file are likely to get lost.

FORTRAN

```
integer function sfend(sd_id)
    integer*4 sd_id
```

sfendacc

SDendaccess

```
intn SDendaccess(int32 sds_id)
```

sds_id	IN:	The data set ID returned from **SDselect**

Purpose — Disposes of a data set ID.

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

Description — When done interacting with a specific data set, this routine should be called to release the internal data structures. This routine should be called once for each call to **SDcreate** or **SDselect**. Failing to call this function may result in loss of data.

FORTRAN

```
integer function sfendacc(sds_id)
    integer*4 sds_id
```

SDfileinfo

sffinfo

SDfileinfo

```
intn SDfileinfo(int32 sd_id, int32 *ndatasets, int32 *nglobal_attr)
```

sd_id	IN:	The SD interface ID returned from **SDstart**
ndatasets	OUT:	Number of data sets in the file
nglobal_attr	OUT:	Number of global attributes in the file

Purpose
Determines the number of data sets and global attributes in a file.

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

Description
SDfileinfo returns the number of currently defined data sets and global attributes in the file. “Global attributes” refer to attributes assigned to the file as opposed to “local attributes” which are assigned to specific data sets. For example, `file_creator_date` would be a global attribute, while `units` would typically be data set specific. Global attributes are created by **SDsetattr** using the *sd_id* rather than an *sds_id*.

FORTRAN

```
integer function sffinfo(sds_id, ndatasets, nglobal_attr)
    integer*4 sds_id, ndatasets, nglobal_attr
```

SDfindattr

```
int32 SDfindattr(int32 s_id, char *attr_name)
```

s_id	IN:	An *sd_id* for global attribute; *sds_id* for a local attribute
attr_name	IN:	Name assigned to the attribute

Purpose — Finds the index for an attribute with a given name.

Return value — Returns the *attr_index* if successful and FAIL(-1) otherwise.

Description — The *attr_index* returned by this function can be passed to **SDattrinfo** and **SDreadattr**. This routine is case sensitive. Wild cards are not allowed in the look up string.

FORTRAN

```
integer function sffattr(s_id, attr_name)
    integer*4       s_id
    character*(*)   attr_name
```

SDgetcal

sfgcal

SDgetcal

```
intn SDgetcal(int32 sds_id, float64 *cal, float64 *cal_err, float64 *offset,
          float64 *offset_err, int32 *number_type)
```

sds_id	IN:	The data set ID returned from **SDselect**
cal	IN:	Calibration factor
cal_err	IN:	Calibration error
offset	IN:	Uncalibrated offset
offset_err	IN:	Uncalibrated offset error
number_type	IN:	Number type of uncalibrated data

Purpose — Gets the calibration information associated with the specified data set.

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

Description — **SDgetcal** reads the calibration record attached to a data set. A calibration record contains four 64-bit floating point values followed by a 32-bit integer, to be interpreted as follows:

cal	calibration factor
cal_err	calibration error
offset	uncalibrated offset
offset_err	uncalibrated offset error
number_type	number type of uncalibrated data

The relationship between a value 'iy' stored in a data set and the actual value is defined as:

y = cal * (iy - 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,
number_type)
    integer*4       sds_id
    real*8          cal, cal_err, offset, offset_err
    integer*4       number_type
```

SDgetdatastrs

```
intn SDgetdatastrs(int32 sds_id, char *label, char *unit, char *format,
          char *coordsys, intn len)
```

sds_id	IN:	The data set ID returned from **SDselect**
label	OUT:	Label that describes the data
unit	OUT:	Unit to be used with the data
format	OUT:	Format to be used in displaying data
coordsys	OUT:	Coordinate system to be used with the data
len	IN:	Maximum length string it is safe to return

Purpose — Returns the data strings associated with the specified data set.

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

Description — **SDgetdatastrs** returns the data strings stored by **SDsetdatastrs**. If a particular string was not stored the first character of the return string will be `'\0'`. Each string buffer is assumed to be at least *len* characters long including the space to hold the `'\0'` string termination character. If a user does not want a string back, `NULL` can be passed for any of the string values.

FORTRAN

```
integer function sfgdtstr(sds_id, label, unit, format,
coordsys, len)
    integer*4       sds_id
    character*(*)   label, unit, format, coordsys
    integer         len
```

SDgetdimid

sfdimid

SDgetdimid

```
int32 SDgetdimid(int32 sds_id, intn dim_number)
```

sds_id	IN:	The data set ID returned from **SDselect**
dim_number	IN:	Number of dimension to get a pointer to

Purpose — Gets the dimension ID associated with the specified dimension in the data set.

Return value — Returns a *dim_id* if successful and `FAIL(-1)` otherwise.

Description — The *dim_id* is required by all other calls that will deal with this particular dimension. Dimensions are zero based so: `0 <= dim_number < rank`.

FORTRAN

```
integer function sfdimid(sds_id, dim_number)
    integer*4 sds_id, dim_number
```

SDgetdimscale

```
intn SDgetdimscale(int32 dim_id, VOIDP data)
```

dim_id	IN:	Dimension ID returned from **SDgetdimid**
data	OUT:	Buffer allocated to hold the scale values for the specified dimension

Purpose — Retrieves the scale values for the specified dimension.

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

Description — It is assumed that the user has already called **SDdiminfo** and has thus allocated sufficient space to hold the values. It is not possible to read only some of the scale values in a hyperslab fashion. **SDgetdimscale** returns all of the scale values stored with the given dimension.

FORTRAN

```
integer function sfgdscale(dim_id, data)
    integer*4       dim_id
    <valid type>    (*)
```

SDgetdimstrs

sfgdmstr

SDgetdimstrs

```
intn SDgetdimstrs(int32 dim_id, char *label, char *unit, char *format, intn len)
```

dim_id	IN:	Dimension ID returned from **SDgetdimid**
label	OUT:	Label that describes this dimension
unit	OUT:	Unit to be used with this dimension
format	OUT:	Format to be used in displaying scale for this dimension
len	IN:	Maximum string length it is safe to return

Purpose — Gets the label, unit, and format strings for a given dimension.

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

Description — This routine returns the data strings stored by **SDsetdimstrs**. If a particular string was not stored, the first character of the return string will be `'\0'`. Each string buffer is assumed to be at least *len* characters long. If a user does not want a string back, `NULL` can be passed for any of the string values.

FORTRAN

```
integer function sfgdmstr(dim_id, label, unit, format, len)
   integer*4      dim_id
   character*(*)  label, unit, format
   integer        len
```

sfgfill

SDgetfillvalue

```
intn SDgetfillvalue(int32 sds_id, VOIDP fill_val)
```

sds_id	IN:	The data set ID returned from **SDselect**
fill_val	OUT:	Space allocated to store the fill value

Purpose — Reads the fill value for the given data set if it exists.

Return value — Returns SUCCEED(0) if successful and FAIL(-1) otherwise.

Description — It is assumed that the number type of the fill value is the same as for the data set.

FORTRAN

```
integer function sfgfill(sds_id, fill_val)
    integer*4       sds_id
    <valid type>    fill_val
```

SDgetinfo

sfginfo

SDgetinfo

```
intn SDgetinfo(int32 sds_id, char *sds_name, int32 *rank, int32 dimsizes[],
          int32 *number_type, int32 *nattrs)
```

sds_id	IN:	The data set ID returned from **SDselect**
sds_name	OUT:	Space to store the name (if any) of the data set
rank	OUT:	Space to store the number of dimensions in the data set
dimsizes	OUT:	Space to store the size of each dimension in the data set
number_type	OUT:	Space to store the number type for the data stored in the data set
nattrs	OUT:	Space to store the number of "netCDF-style" attributes for this data set

Purpose — Gets the name, rank, dimension sizes, number type and attribute count for the specified data set.

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

Description — Returns basic information about a given data set. All fields must be provided. The data set name can be at most `MAX_NC_NAME` characters long and the rank of the data set is limited to `MAX_VAR_DIMS`.

FORTRAN

```
integer function sfginfo(sds_id, sds_name, rank, dimsizes,
number_type, nattrs)
    character*(*)  sds_name
    integer*4      sds_id, rank, dimsizes(*)
    integer*4      number_type, nattrs
```

SDgetrange

```
intn SDgetrange(int32 sds_id, VOIDP max, VOIDP min)
```

sds_id	IN:	The data set ID returned from **SDselect**
max	OUT:	High value in range
min	OUT:	Low value in range

Purpose — Reads the maximum and minimum values as they are stored with the scientific data set.

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

Description — The *max* and *min* must be set via a call to **SDsetrange**. They are not automatically stored when a data set is written to a file.

Since the *max* and *min* values are supposed to relate to the data itself, it is assumed that they are of the same number type as the data itself. One implication of this is that in the C version of **SDgetrange** the arguments are pointers, rather than simple variables, whereas in the FORTRAN version they are simple variables of the same number type as the data array.

FORTRAN

```
integer function sfgrange(sds_id, max, min)
    integer*4       sds_id
    <valid type>    max
    <valid type>    min
```

SDidtoref

sfid2ref

SDidtoref

```
int32 SDidtoref(int32 sds_id)
```

sds_id	IN:	The data set ID returned from **SDselect**

Purpose — Uses a data set ID to determine the reference number assigned to the specified data set.

Return value — Returns *sds_ref* if successful and FAIL(-1) otherwise.

Description — This 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*4 sds_id
```

SDiscoordvar

```
intn SDiscoordvar(int32 sds_id)
```

sds_id	IN:	The data set ID returned from **SDselect**

Purpose — Determines if a data set is a coordinate variable

Return value — Returns TRUE if the data set is a coordinate variable, and FALSE otherwise.

Description — This routine can be used to tell whether a given data set is a "coordinate variable" or not. Coordinate variables are created to store metadata associated with dimensions. Due to netCDF compatibility, coordinate variables are also considered data sets which sometimes is a source of confusion.

FORTRAN

```
integer function sfiscvar(sds_id)
    integer*4 sds_id
```

SDnametoindex

sfn2index

SDnametoindex

```
int32 SDnametoindex(int32 sd_id, char *sds_name)
```

sd_id	IN:	The SD interface ID returned from **SDstart**
sds_name	IN:	Name of the data set to index

Purpose — Uses the name of a scientific data set to determine the index assigned to it.

Return value — Returns an *sds_index* if successful and FAIL(-1) otherwise.

Description — The *sds_index* can be passed to **SDselect** to return an *sds_id* for the named data set. This routine is case sensitive and will not accept wild cards. In addition, this routine will only return the *sds_index* for the first data set with the given name.

FORTRAN

```
integer function sfn2index(sd_id, sds_name)
   integer*4      sd_id
   character*(*)  sds_name
```

SDreadattr

```
intn SDreadattr(int32 s_id, int32 attr_index, VOIDP data)
```

s_id	IN:	*sd_id* if reading a global attribute; an *sds_id* if reading a local attribute
attr_index	IN:	Index of the attribute to read
data	OUT:	Buffer allocated to hold attribute values

Purpose — Reads the values of an attribute.

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

Description — It's assumed that the user has called **`SDattrinfo`** and that the buffer is large enough to handle the data. If an attribute has multiple values stored for it, this routine will return all of them. It is not possible to read a subset of attribute values.

FORTRAN

```
integer function sfrattr(s_id, attr_index, data)
    integer*4       s_id
    integer*4       attr_index
    <valid type>    data(*)
```

SDreaddata

sfrdata

SDreaddata

```
intn SDreaddata(int32 sds_id, int32 start[], int32 stride[], int32 edge[],
          VOIDP buffer)
```

sds_id	IN:	The data set ID returned from **SDselect**
start	IN:	Array specifying the starting location
stride	IN:	Array specifying the number of values to skip along each dimension
edge	IN:	Array specifying the number of values to read along each dimension
buffer	OUT:	Buffer to store the data

Purpose
Reads a hyperslab of data from a data set.

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

Description
The array *start* gives the multi-dimensional index of the starting corner of the hyperslab to read. The values are zero based.

The array *edge* gives the number of values to read along each dimension of the hyperslab.

The array *stride* allows for sub-sampling along each dimension. If a stride value is specified for a dimension, that many values will be skipped over when reading along that dimension. Specifying *stride*=NULL reads contiguous data. No matter what stride value is provided, data is always placed contiguously in buffer.

FORTRAN

```
integer function sfrdata(sds_id, start, stride, edge, buffer)
   integer*4      sds_id, start(*), stride(*), edge(*)
   <valid type>   buffer(*)
```

sfref2index

SDreftoindex

```
int32 SDreftoindex(int32 sd_id, int32 sds_ref)
```

sd_id	IN:	The SD interface ID returned from **SDstart**
sds_ref	IN:	Reference number for the specified data set

Purpose — Uses the reference number to determine the index assigned to a data set.

Return value — Returns an *sds_index* if successful and FAIL(-1) otherwise.

Description — This *sds_index* can be passed to **SDselect** to return a data set ID (*sds_id*).

FORTRAN

```
integer function sfref2index(sd_id, sds_ref)
    integer*4 sd_id, sds_ref
```

SDselect

sfselect

SDselect

```
int32 SDselect(int32 sd_id, int32 index)
```

sd_id	IN:	The SD interface ID returned from **SDstart**
sds_index	IN:	The position of the data set in the file

Purpose — Gets the sds_id for a specific data set.

Return value — Returns the *sds_id* if successful and `FAIL(-1)` otherwise.

Description — To get an *sds_id* for the Nth data set, use N as the index. N must be a number greater than or equal to 0 and less than the total 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 function **SDnametoindex** can be used to find the index of a data set if its name is known.

The integration with netCDF has required that dimension metadata be stored as "coordinate variables." This fact sometimes causes problems when calling **SDselect** since the coordinate variables can lead to different data set ordering than is expected. In situations such as these, users should use the routine **SDiscoordvar** to determine if a given data set is a coordinate variable or not.

FORTRAN

```
integer function sfselect(sd_id, index)
    integer*4 sd_id, index
```

sfsattr

SDsetattr

```
intn SDsetattr(int32 s_id, char *attr_name, int32 num_type, int32 count,
          VOIDP values)
```

s_id	IN:	A *sd_id* to create a global attribute; a *sds_id* to create a local attribute
attr_name	IN:	Name to assign to the attribute
num_type	IN:	Number type for the values in the attribute
count	IN:	Total number of values to store in the attribute
values	IN:	Data values to store in the attribute

Purpose Defines a new type of attribute for the given variable.

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

Description This routine provides a generic way for users to define metadata. It implements the `label=value` data abstraction.

The *attr_name* argument can be any ASCII string for which one or more values may be stored. If more than one value is stored, all values must have the same number type. If an attribute with the given scope and name exists, it will be over written.

If an *sd_id* is given instead of an *sds_id*, a global attribute is created which applies to the whole file. Global attributes refer to all objects in a file as opposed to specific data sets. For example, `file_creator_date` is a global attribute, where as `units` is typically considered to be data set specific.

Example Store a `valid_range` attribute for a data set.

```
int32 range[2];
int32 sds_id;
int32 status;

...
sds_id = SDcreate...;
...

range[0] = 5;
range[1] = 100;
status = SDsetattr(sds_id, "valid_range", DFNT_INT32, 2, range);
```

FORTRAN

```
integer function sfsattr(s_id, attr_name, num_type, count, values)
   character*(*)  attr_name, values
   integer*4      s_id, num_type, count
   <valid type>   values(*)
```

SDsetcal

sfscal

SDsetcal

```
intn SDsetcal(int32 sds_id, float64 cal, float64 cal_err, float64 offset,
          float64 offset_err, int32 number_type)
```

sds_id	IN:	The data set ID returned from **SDselect**
cal	IN:	Calibration factor
cal_err	IN:	Calibration error
offset	IN:	Uncalibrated offset
offset_err	IN:	Uncalibrated offset error
number_type	IN:	Number type of uncalibrated data

Purpose Sets the calibration information.

Return value Returns `SUCCEED(0)` if successful and `FAIL(-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
number_type	number type of uncalibrated data

The relationship between a value 'iy' stored in a data set and the actual value is defined as:

y = cal * (iy - 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.

SDsetcal works like other **SDset*** routines, with one exception: the calibration information is automatically cleared after a call to **SDreaddata** or **SDwritedata**. Hence, **SDsetcal** must be called anew for each data set that is to be written.

FORTRAN

```
integer function sfscal(sds_id, cal, cal_err, offset, offset_err,
number_type)
    integer*4       sds_id, number_type
    real*8          cal, cal_err, offset, offset_err
```

SDsetdatastrs

```
intn SDsetdatastrs(int32 sds_id, char *label, char *unit, char *format,
          char *coordsys)
```

sds_id	IN:	The data set ID returned from **SDselect**
label	IN:	Label that describes the data
unit	IN:	Unit to be used with the data
format	IN:	Format to be used in displaying the data
coordsys	IN:	Coordinate system to be used with the data

Purpose — Sets the label, unit, format, and coordinate system strings for a given data set.

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

Description — This routine can be used to set the label, unit, format, and coordinate system strings for a given data set. `NULL` can be passed for any strings the user does not want to set.

FORTRAN

```
integer function sfsdtstr(sds_id, label, unit, format,
coordsys)
    integer*4       sds_id
    character*(*)   label, unit, format, coordsys
```

SDsetdimname

sfsdmname

SDsetdimname

```
intn SDsetdimname(int32 dim_id, char *dim_name)
```

dim_id	IN:	Dimension ID returned from **SDgetdimid**
dim_name	IN:	ASCII string to name dimension

Purpose — Assigns a name to the specified dimension.

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

Description — 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. Naming dimensions is optional but encouraged. Dimensions that are not explicitly named by the user will have generated names used as place holders.

FORTRAN

```
integer function sfsdmname(dim_id, dim_name)
    integer*4      dim_id
    character*(*)  dim_name
```

sfsdscale

SDsetdimscale

```
intn SDsetdimscale(int32 dim_id, int32 count, int32 number_type, VOIDP data)
```

dim_id	IN:	Dimension ID returned from **SDgetdimid**
count	IN:	Total number of values along the specified dimension
number_type	IN:	Number type for the values along the specified dimension
data	IN:	Value for each increment along the specified dimension

Purpose — Stores the values of the specified dimension.

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

Description — Users should note that it is possible to store values for a dimension without having to name it. Even though its redundant we have left the *count* argument in for backward compatibility. Note that there is now a *number_type* parameter so the dimension scales no longer have to be the same number type as the data set.

FORTRAN

```
integer function sfsdscale(dim_id, count, number_type, data)
    integer*4       dim_id
    integer*4       count, number_type
    <valid type>    data(*)
```

SDsetdimstrs

sfsdmstr

SDsetdimstrs

```
intn SDsetdimstrs(int32 dim_id, char *label, char *unit, char *format)
```

dim_id	IN:	Dimension ID returned from **SDgetdimid**
label	IN:	Label that describes this dimension
unit	IN:	Unit to be used with this dimension
format	IN:	Format to be used to display scale

Purpose — Sets the label, unit, and format strings for a given dimension.

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

Description — **SDsetdimstrs** sets the label, unit, and format specifications for a dimension and its scale. If `NULL` is passed for a string, that string will not be written.

FORTRAN

```
integer function sfsdmstr(dim_id, label, unit, format)
   integer*4      dim_id
   character*(*)  label, unit, format
```

SDsetexternalfile

```
intn SDsetexternalfile(int32 sds_id, char *filename, int32 offset)
```

sds_id	IN:	The data set ID returned from **SDselect**
filename	IN:	Name of the external file
offset	IN:	Byte count from the beginning of the external file to where the data starts

Purpose — Stores data in an external file.

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

Description — **SDsetexternalfile** allows users to move the data values for a data set into an external data file. Only the data set values can be stored externally, all other data must stay in the original file.

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 transported with the "original" file. The *offset* is the number of bytes from the beginning of the external file to where the external data begins. This routine can only be called on HDF 3.3 files (i.e. calling on a netCDF file that was opened with the multi-file interface will fail).

FORTRAN — None

SDsetfillvalue

sfsfill

SDsetfillvalue

```
intn SDsetfillvalue(int32 sds_id, VOIDP fill_val)
```

sds_id	IN:	The data set ID returned from **SDselect**
fill_val	IN:	Pointer to the fill value to be used

Purpose — Sets the fill value for the given data set.

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

Description — This routine sets the fill value to be used for the specified data set. Since data can be written in non-contiguous hyperslabs, it is possible that not all of the locations in a given data set will be written to. The *fill_val* is placed into all unwritten locations. It is assumed that the fill value has the same number type as the data set.

To avoid consistency problems, this routine should be called before any data is written to the data set. In addition, once set, the fill value should not be changed.

FORTRAN

```
integer function sfsfill(sds_id, fill_val)
   integer*4      sds_id
   <valid type>   fill_val
```

sfsrange

SDsetrange

```
intn SDsetrange(int32 sds_id, VOIDP max, VOIDP min)
```

sds_id	IN:	The data set ID returned from **SDselect**
max	IN:	High value in range
min	IN:	Low value in range

Purpose — Sets the "valid" max and min values for the given data set.

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

Description — Since *max* and *min* are supposed to relate to the data itself, it is assumed that the number type for the *max*/*min* values are the same as the number type of the data. One implication of this is that in the C version of **SDsetrange** the arguments are pointers, rather than simple variables, whereas in the FORTRAN version they are simple variables of the same type as the data array.

This routine does not compute the maximum and minimum values; it merely stores the values it is given. As a result, the maximum and minimum values may not always reflect the actual maximum and minimum values in the data array.

FORTRAN

```
integer function sfsrange(sds_id, max, min)
    integer*4       sds_id
    <valid type>    max, min
```

SDstart

sfstart

SDstart

```
int32 SDstart(char *filename, int32 access_mode)
```

filename	IN:	Name of the HDF file
access_mode	IN:	Recognized access code

Purpose Opens the HDF file and initializes the SD interface.

Return value Returns an *sd_id* if successful and FAIL (-1) otherwise.

Description This routine opens a file and returns an *sd_id*. This routine must be called for each file before any other SD calls can be made on that file. The *access_mode* is one of the following:

DFACC_RDONLY
DFACC_RDWR
DFACC_CREATE (destroy existing contents and then rewrite)

The file can be any one of the following:

XDR-based netCDF file
"old-style" DFSD file
"new-style" SD file

If *access_mode* DFACC_CREATE is used "new-style" SD files will be created.

The type of ID returned by **SDstart** is currently not the same as the ID returned by **Hopen**. As a result, *sd_id*s are not understood by other HDF interfaces and *h_id*s are not recognized by the SD interface.

To mix SD calls and other HDF library calls, use **SDstart** and **Hopen** on the same file. **SDstart** must precede all SD calls, and **Hopen** must proceed all other HDF function calls. To terminate access to the file, use both **SDend** and **Hclose** to dispose of the *sd_id* and the *h_id*.

FORTRAN

```
integer function sfstart(filename, access_mode)
    character*(*)  filename
    integer*4      access_mode
```

SDwritedata

```
intn SDwritedata(int32 sds_id, int32 start[], int32 stride[], int32 edge[],
          VOIDP data)
```

sds_id	IN:	The data set ID returned from **SDselect**
start	IN:	Array specifying the starting location
stride	IN:	Array specifying the number of values to skip along each dimension
edge	IN:	Array specifying the number of values to read along each dimension
data	IN:	Values to be written to the data set

Purpose — Writes a hyperslab of data for a data set.

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

Description — The array *start* gives the multi-dimensional index of the starting corner of the hyperslab to write. The values are zero based.

The array *edge* gives the number of values to write along each dimension of the hyperslab.

The array *stride* allows for sub-sampling along each dimension. If a stride value is specified for a dimension, that many values will be skipped over when writing along that dimension. Specifying *stride*=NULL writes contiguous data. No matter what stride value is provided, data is always read contiguously from *data*.

FORTRAN

```
integer function sfwdata(sds_id, start, stride, edge, data)
    integer*4       sds_id
    integer         start(*), stride(*), edge(*)
    <valid type>    data(*)
```