SDattrinfo/sfgainfo

intn SDattrinfo(int32 *obj_id,* int32 *attr_index*, char **attr_name*, int32 **data_type*, 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
data_type	OUT:	Data 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, data type, and number of values of the attribute specified by its index, *attr_index,* and stores them in the parameters *attr_name*, *data_type*, 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 *data_type* can be found in Table 1A of Section I of this manual.

FORTRAN

```
integer function sfgainfo(obj_id, attr_index, attr_name,
                                data_type, count)

character*(*) attr_name

integer obj_id, attr_index, data_type, count
```

SDcreate/sfcreate

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

sd_id	IN:	SD interface identifier returned by **SDstart**
name	IN:	Name of the data set
data_type	IN:	Data 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 specified by the parameter *name*, the values of the data type specified by parameter *data_type*, the number of dimensions specified by the parameter *rank*, and the dimension sizes specified by the array *dimsizes*.

Once a data set has been created, it is not possible to change its name, data 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 "Data Set" will be generated. If the length of the name specified by the *name* parameter is longer than `64` characters, then the name will be truncated to `64` characters.

The calling program must ensure that the length of the *dimsizes* array is the value of the *rank* parameter, which is between 1 and `MAX_VAR_DIMS` (or `32`).

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

FORTRAN

```
integer function sfcreate(sd_id, name, data_type, rank,
                                    dimsizes)

character*(*) name

integer sd_id, data_type, rank, dimsizes(*)
```

SDdiminfo/sfgdinfo

intn SDdiminfo(int32 *dim_id*, char **name*, int32 **size*, int32 **data_type*, int32 **num_attrs*)

dim_id	IN:	Dimension identifier returned by **SDgetdimid**
name	OUT:	Name of the dimension
size	OUT:	Size of the dimension
data_type	OUT:	Data 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, data type, and number of values of the dimension specified by the parameter *dim_id*, and stores them in the parameters *name*, *size*, *data_type*, 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 *data_type* parameter will contain the data type. Valid data types can be found in Table 1A of Section I of this manual. If no scale information has been stored for this dimension, the value returned in the *data_type* 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, data_type,
                                    num_attrs)

character*(*) name

integer dim_id, size, data_type, 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
```

SDgetcal/sfgcal

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

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
data_type	OUT:	Data 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
`data_type`	data 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,
                        data_type)

integer sds_id, data_type

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 (Fortran)	*cdef* Structure Element Filled with the Chunk's Dimensions
Not chunked	`HDF_NONE (-1)`	None
Chunked	`HDF_CHUNK (0)`	`cdef.chunk_lengths[]`

Type of Data Set	Values of *flag* in C (Fortran)	*cdef* Structure Element Filled with the Chunk's Dimensions
Chunked and compressed with either the run-length encoding (RLE), Skipping Huffman or GZIP 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
```

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 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*.

SDdiminfo should be used to determine whether a scale has been set for the dimension, i.e., that the dimension scale data type is a valid HDF data type (not `0`). Also use **SDdiminfo** to obtain the number of scale values for space allocation before calling **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 Section 3.10 of 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
```

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 data type of the fill value is the same as that of the data set.

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 **data_type*, 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
data_type	OUT:	Data 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, data 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, data type, and number of attributes of the data set identified by *sds_id*, and stores them in the parameters *sds_name*, *rank*, *dimsizes*, *data_type*, and *num_attrs*, respectively.

The buffer *sds_name* can have at most `64` characters. 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 of the *rank* parameter 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,
                         data_type, num_attrs)

character*(*) sds_name

integer sds_id, rank, dimsizes(*)

integer data_type, num_attrs
```

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 data 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
```

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 number if 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
```

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 the *HDF User's Guide*, Chapter 3, titled *SD Scientific Data Sets (SD API)*, 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 "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.

FORTRAN

```
integer function sfn2index(sd_id, sds_name)

integer sd_id

character*(*) sds_name
```

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 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 of the *HDF User's Guide*, titled *Scientific Data Sets (SD API)*, 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, **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 of the HDF User's Manual, titled *Scientific Data Sets (SD API)* and Chapter 13 of the HDF User's Manual, titled *HDF Performance Issues*.

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.

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
```

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.

FORTRAN

```
integer function sfselect(sd_id, sds_index)

integer sd_id, sds_index
```

SDsetattr/sfsnatt/sfscatt

intn SDsetattr(int32 *obj_id*, char **attr_name*, int32 *data_type*, 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
data_type	IN:	Data 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*, data type, *data_type*, 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 **SDcreate**, a data set identifier (*sds_id*), returned by **SDselect**, or a dimension identifier (*dim_id*), returned by **SDgetdimid**.

If an SD interface identifier (*sd_id*) is specified as the *obj_id* parameter, a global attribute is created which applies to all objects in the file. If a data set identifier (*sds_id*) is specified as the *obj_id* parameter, an attribute is attached to the specified data set. If a dimension identifier (*dim_id*) is specified as the *obj_id* parameter, an attribute is attached to the specified dimension.

The *attr_name* argument can be any ASCII string.

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

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

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, data_type, count,
                                    values)

integer obj_id, data_type, count
```

```
character*(*) attr_name

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

```
integer function sfscatt(obj_id, attr_name, data_type, count,
                         values)

integer obj_id, data_type, 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 *data_type*)

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
data_type	IN:	Data 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
`data_type`	Data 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,
                        data_type)
```

```
integer sds_id, data_type

real*8 cal, cal_err, offset, offset_err
```

SDsetchunk/sfschnk

intn s(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_flag	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 or NBIT compression methods. Valid values of *flag* are HDF_CHUNK for a chunked data set, HDF_CHUNK | HDF_COMP for a chunked data set compressed with RLE, Skipping Huffman and GZIP compression methods, and HDF_CHUNK | HDF_NBIT for a chunked NBIT-compressed data set.

Chunking and compression information is 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;
          } 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 data set is chunked and compressed using RLE, Skipping Huffman 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 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`). Valid compression types are: `COMP_CODE_RLE` for RLE, `COMP_CODE_SKPHUFF` for Skipping Huffman, `COMP_CODE_DEFLATE` for GZIP compression.

For Skipping Huffman and GZIP compression parameters are passed in corresponding fields of the structure `cinfo`. Specify skipping size for Skipping Huffman compression in the field `cdef.comp.cinfo.skphuff.skp_size`. Specify deflate level for GZIP compression in the field `cdef.comp.cinfo.deflate_level`. Valid values of deflate levels are integers between 1 and 9 inclusive.

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

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 uncompressed data
`COMP_CODE_RLE` (or 1) for data compressed using the RLE compression algorithm
`COMP_CODE_NBIT` (or 2) for data compressed using the NBIT compression algorithm
`COMP_CODE_SKPHUFF` (or 3) for data compressed using the Skipping Huffman compression algorithm
`COMP_CODE_DEFLATE` (or 4) for data compressed using the GZIP compression algorithm

The *comp_prm*(1) parameter specifies the skipping size for the Skipping Huffman compression method and the deflate level for the GZIP compression method.

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.

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 13 of the HDF User's Guide, titled *HDF Performance Issues*, 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** compresses the data set data at the time it is called, not 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 `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, or `COMP_CODE_NONE` (or 0) for no 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 a gzip encoded dataset */
    /* how hard to work when compressing data */
    intn level;
  } deflate;

} comp_info;
```

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

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

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 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.

Naming dimensions is optional but encouraged.

The length of the parameter *dim_name* can be at most 64 characters.

FORTRAN

```
integer function sfsdmname(dim_id, dim_name)

integer dim_id

character*(*) dim_name
```

SDsetdimscale/sfsdscale

intn SDsetdimscale(int32 *dim_id*, int32 *count*, int32 *data_type*, VOIDP *data*)

dim_id	IN:	Dimension identifier returned by **SDgetdimid**
count	IN:	Total number of values along the dimension
data_type	IN:	Data 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 *data_type*, the dimension scales need not have the same data type as the data set.

FORTRAN

```
integer function sfsdscale(dim_id, count, data_type, data)

integer dim_id, count, data_type

<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 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 data type as the data set.

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 `SUCCEED` (or `0`) 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 data 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_bit* parameter 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 data type for the maximum and minimum range values are the same as the data 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.

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 of the HDF User's Guide, titled *Scientific Data Sets (SD API)*, 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**).

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 be issues presented in the section on chunking in Chapter 3 of the HDF User's Manual, titled *Scientific Data Sets (SD API)* and Chapter 13 of the HDF User's Manual, titled *HDF Performance Issues*.

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.

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(*)
```