SDattrinfo/sfgainfo

intn SDattrinfo(int32 [*file*, *sds*, *dim*]_*id*, int32 *attr_index*, char **attr_name*, int32 **data_type*, int32 **count*)

[*file*, *sds*, *dim*]_*id*	IN:	Identifier of the object the attribute is to be attached to: an *file_id* for a file, an *sds_id* for an SDS or a *dim_id* for a dimension
attr_index	IN:	Index of the attribute to read
attr_name	OUT:	Name assigned to the attribute/dataset/dimension
data_type	OUT:	Data type of the attribute values
count	OUT:	Total number of values in the specified attribute

Purpose Retrieves information about a global or local attribute.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-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**(*data_type*) bytes long.

FORTRAN

```
integer function sfgainfo([file, sds, dim]_id, attr_index,
     attr_name, data_type, count)

character* (*) attr_name
integer [file, sds, dim]_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 from **SDstart**
name	IN:	ASCII string defining a variable name
data_type	IN:	Data type for the values in the dataset
rank	IN:	Number of dimensions in the dataset
dimsizes	IN:	Size of each dimension

Purpose Creates a new dataset.

Return value Returns the *sds_id* if successful and `FAIL` (or `-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 dataset has been created, it is impossible to change its name, data type, or rank. However, it is possible to create a dataset 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 dataset, 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 to this dimension at will. In Fortran-77, *dimsizes*(*rank*) is the only dimension that can be unlimited. This is useful when the eventual size of a dataset is not known at creation time.

Valid values for *data_type* are prefaced by DFNT_. The following are valid symbolic names and their data types:

32-bit float	DFNT_FLOAT32	5
64-bit float	DFNT_FLOAT64	6
8-bit signed int	DFNT_INT8	20
8-bit unsigned int	DFNT_UINT8	21
16-bit signed int	DFNT_INT16	22
16-bit unsigned int	DFNT_UINT16	23
32-bit signed int	DFNT_INT32	24
32-bit unsigned int	DFNT_UINT32	25

8-bit unsigned character DFNT_CHAR84

MAX_VAR_DIMS is the maximum rank a dataset can have.

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 **count*, int32 **data_type*, int32 **nattrs*)

dim_id	IN:	Dimension identifier returned from **SDgetdimid**
name	OUT:	Array to retrieve dimension name
count	OUT:	Size of the dimension
data_type	OUT:	Data type of the data stored in the dataset
nattrs	OUT:	Attribute count assigned to the dimension's coordinate variable

Purpose Retrieves information about a dimension.

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

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

The *count* argument is set to `SD_UNLIMITED` for unlimited dimensions. To get the number of records of an unlimited dimension, use **SDgetinfo**.

If "label, unit, format" metadata (or "LUF" metadata) is assigned to a dimenstion, it will be "promoted" by the SD interface to be a variable. If the target dimension specfied by *dim_id* is a varaible, the value returned in the *data_type* argument will be `0`.

FORTRAN

```
integer function sfgdinfo(dim_id, name, count, data_type,
                               nattrs)

character* (*) name
integer dim_id, count, data_type, nattrs
```

SDend/sfend

intn SDend(int32 *sd_id*)

sd_id	IN:	SD interface identifier returned from **SDstart**

Purpose Closes the file and frees memory allocated by the library.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-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 data in-core are likely not to be flushed to the file.

FORTRAN

```
integer function sfend(sd_id)

integer sd_id
```

SDendaccess/sfendacc

intn SDendaccess(int32 *sds_id*)

sds_id	IN:	Dataset identifier returned from **SDselect**

Purpose Disposes of a dataset identifier.

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

Description When done interacting with a specific dataset, 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 sds_id
```

SDfileinfo/sffinfo

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

sd_id	IN:	SD interface identifier returned from **SDstart**
ndatasets	OUT:	Number of datasets in the file
nglobal_attr	OUT:	Number of global attributes in the file

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

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

Description **SDfileinfo** returns the number of currently defined datasets 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 datasets. For example, "file_creator_date" would be a global attribute, while "units" would typically be dataset-specific. Global attributes are created by **SDsetattr** using the *sd_id* rather than an *sds_id*.

FORTRAN

```
integer function sffinfo(sd_id, ndatasets, nglobal_attr)

integer sd_id, ndatasets, nglobal_attr
```

SDfindattr/sffattr

int32 SDfindattr(int32 [*file*, *sds*, *dim*]_*id*, char **attr_name*)

[*file*, *sds*, *dim*]_*id*	IN:	Identifier of the object the attribute is to be attached to: an *file_id* for a file, an *sds_id* for an SDS or a *dim_id* for a dimension
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` (or `-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 *attr_name* parameter.

FORTRAN

```
integer function sffattr([file, sds, dim]_id, attr_name)

integer [file, sds, dim]_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:	Dataset identifier returned from **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 the specified dataset.

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

Description **SDgetcal** reads the calibration record attached to a dataset. 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`
`data_type`	`data type of uncalibrated data`

The relationship between a value `iy` stored in a dataset 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, data_type)

integer sds_id, data_type
real*8 cal, cal_err, offset, offset_err
```

SDgetchunkinfo

intn SDgetchunkinfo(int32 *sds_id*, HDF_CHUNK_DEF **cdef*, int32 **flags*)

sds_id	IN:	Dataset identifier returned from **SDselect**
cdef	OUT:	Union structure containing information about the chunks in the SDS
flags	OUT:	Flags determining routine behavior

Purpose Obtains chunking information about a scientific dataset.

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

Description A pointer to a `chunk_lengths` array (defined in the `comp_info` union pointed to by *cdef*) containing chunk dimension size information will be returned if the value of `flags` is `HDF_CHUNK`, and a pointer to a `comp_info` structure containing the dimension size of the chunks and compression information will be returned if the value of `flags` is `HDF_CHUNK` bitwise-OR'ed with `HDF_COMP`.

For the definition of the `HDF_CHUNK_DEF` union, refer to the Reference Manual page for **SDsetchunk**.

SDgetchunkinfo can also be used to determine if the target SDS is not chunked. In this case, the value of `flags` is set to `HDF_NONE` and the *cdef* parameter is ignored

A `NULL` value can also be passed in as the `cdef` parameter if chunking information is not desired.

SDgetdatastrs/sfgdtstr

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

sds_id	IN:	Dataset identifier returned from **SDselect**
label	OUT:	Label describing 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 dataset.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-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 sds_id, len
character* (*) label, unit, format, coordsys
```

SDgetdimid/sfdimid

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

sds_id	IN:	Dataset identifier returned from **SDselect**
dim_number	IN:	Number of dimensions

Purpose	Retrieves the dimension identifier associated with the specified dimension in the dataset.
Return valu	Returns a *dim_id* if successful and `FAIL` (or `-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 sds_id, dim_number`

SDgetdimscale/sfgdscale

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

dim_id	IN:	Dimension identifier returned from **SDgetdimid**
data	OUT:	Buffer for the scale values for the specified dimension

Purpose Retrieves the scale values for the specified dimension.

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

Description It is assumed that the user has already called **SDdiminfo** and has thus allocated sufficient space to hold the values. **SDdiminfo** is used to check if a scale has been set for the target dimension. If the data type parameter value returned by **SDdiminfo** is `0`, no scale has been set and **SDgetdimscale** should not be called.

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, data)

integer dim_id
<valid numeric data type> data
```

SDgetdimstrs/sfgdmstr

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

dim_id	IN:	Dimension identifier 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 Retrieves the label, unit, and format strings for a given dimension.

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

Description This routine returns the dimension 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 returned, `NULL` can be passed for any of the string values.

FORTRAN

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

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

SDgetfillvalue/sfgfill/sfgcfill

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

sds_id	IN:	Dataset identifier returned from **SDselect**
fill_val	OUT:	Buffer for the returned fill value

Purpose Reads the fill value for the given dataset if it exists.

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

Description It is assumed that the data type of the fill value is the same as for the dataset.

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_val)

integer sds_id
<valid numeric data type> fill_val


integer function sfgcfill(sds_id, fill_val)

integer sds_id
character* (*) fill_val
```

SDgetinfo/sfginfo

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

sds_id	IN:	Dataset identifier returned from **SDselect**
sds_name	OUT:	Buffer for the name, if any, of the dataset
rank	OUT:	Buffer for the number of dimensions in the dataset
dimsizes	OUT:	Buffer for the size of each dimension in the dataset
data_type	OUT:	Buffer for the data type for the data stored in the dataset
nattrs	OUT:	Buffer for the number of "netCDF-style" attributes for this dataset

Purpose Retrieves the name, rank, dimension sizes, data type and attribute count for the specified dataset.

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

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

In the case of unlimited dimensions, the *dimsizes*[0] argument returns the number of records in the dimension.

FORTRAN

```
integer function sfginfo(sds_id, sds_name, rank, dimsizes,
    data_type, nattrs)

character* (*) sds_name
integer sds_id, rank, dimsizes(*)
integer data_type, nattrs
```

SDgetrange/sfgrange

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

sds_id	IN:	Dataset identifier returned from **SDselect**
max	OUT:	Highest value in the range
min	OUT:	Lowest value in the range

Purpose Retrieves the maximum and minimum values as they are stored with the scientific dataset.

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

Description The maximum and minimum values must be set via a call to **SDsetrange**. They are not automatically stored when a dataset is written to a file. As the max imumand minimum values are supposed to relate to the data itself, it is assumed that they are of the same data 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-77 version they are simple variables of the same data type as the data array.

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:	Dataset identifier returned from **SDselect**

Purpose Retrieves the reference number assigned to the specified dataset.

Return value Returns sds_ref if successful and `FAIL` (or `-1`) otherwise.

Description The specified reference number can be used to add the dataset to a vgroup as well as a means of using the HDF annotations interface to annotate the dataset.

FORTRAN

```
integer function sfid2ref(sds_id)

integer sds_id
```

SDiscoordvar/sfiscvar

intn SDiscoordvar(int32 *sds_id*)

sds_id	IN:	Dataset identifier returned from **SDselect**

Purpose Determines if a dataset is a coordinate variable

Return value Returns `TRUE` if the dataset is a coordinate variable, and `FALSE` otherwise.

Description Coordinate variables are created to store metadata associated with dimensions. Due to netCDF compatibility, coordinate variables are also considered datasets which sometimes is a source of confusion.

FORTRAN

```
integer function sfiscvar(sds_id)

integer sds_id
```

SDisdimval_bwcomp/sfisdmvc

intn SDisdimval_bwcomp(int32 *dim_id*)

dim_id	IN:	Dimension identifier returned from **SDgetdimid**

Purpose Determines whether the specified dimension *will have* the old and new representations or the new representation only.

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 **SDstart** reads dimension records into memory. The compatiblity mode of each dimesion is decided by the existance of the dimension vdata of class "DimVal0.0" in the dimension vgroup for that dimension.

If "DimVal0.0" vdata doesn't exist in that dimension vgroup, it will be flagged 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

int32 SDisrecord(int32 *sds_id*)

sds_id	IN:	SDS identifier returned from **SDselect**

Purpose Determines if a dataset is a record variable

Return value Returns `TRUE` if the dataset is a record variable, and `FALSE` otherwise.

SDnametoindex/sfn2index

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

sd_id	IN:	SD interface identifier returned from **SDstart**
sds_name	IN:	Name of the dataset to index

Purpose Determines the index assigned to the scientific dataset defined by *sds_name*.

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

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

FORTRAN

```
integer function sfn2index(sd_id, sds_name)

integer sd_id
character* (*) sds_name
```

SDreadattr/sfrnatt/sfrcatt

intn SDreadattr(int32 [*file*, *sds*, *dim*]_*id*, int32 *attr_index*, VOIDP *data*)

[*file*, *sds*, *dim*]_*id*	IN:	Identifier of the object the attribute is to be attached to: an *file_id* for a file, an *sds_id* for an SDS or a *dim_id* for a dimension
attr_index	IN:	Index of the attribute to be read
data	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 It's assumed that the user has called **SDattrinfo** and that the buffer is large enough to store 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.

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.

The index returned as the *attr_index* argument is one-based.

FORTRAN

```
integer function sfrnatt([file, sds, dim]_id, attr_index, data)

integer [file, sds, dim]_id, attr_index
<valid numeric data> data


integer function sfrcatt([file, sds, dim]_id, attr_index, data)

integer [file, sds, dim]_id, attr_index
character* (*) data
```

SDreadchunk

intn SDreadchunk(int32 *sds_id*, int32 **origin*, VOIDP *datap*)

sds_id	IN:	SD interface identifier returned from **SDstart**
origin	IN:	Origin of the chunk to be read
datap	OUT:	Buffer for the returned chunk data

Purpose Reads data from a chunked scientific dataset.

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

Description **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 SDS.

Also, **SDreadchunk** is written specifically for chunked SDSs and and doesn't have the overhead of the additional functionality supported by the **SDreaddata** routine - therefore, it is much faster than **SDreaddata**.

SDreadchunk will return `FAIL` when an attempt is made to use it to read from a non-chunked SDS.

SDreaddata/sfrdata/sfrcdata

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

sds_id	IN:	Dataset identifier 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 for the data

Purpose Reads a hyperslab of data from a dataset.

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

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

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

The *stride* array 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` in the C interface or *stride* = `1` in either interface specifies contiguous reading of data. If the *stride* values are set to `0`, **SDreaddata** returns `FAIL` (or `-1`). No matter what stride value is provided, data is always placed contiguously in buffer.

If the SDS specified by *sds_id* contains no data, **SDreaddata** returns `0`.

When reading data from a "chunked" SDS using **SDreaddata**, consideration should be given to be issues presented in the section on chunking in Chapter 3 of the HDF User's Manual, tited *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> 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 from **SDstart**
sds_ref	IN:	Reference number for the specified dataset

Purpose Determines the index assigned to a scientific dataset given the specified reference number.

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

Description The value of *sds_index* can be passed to **SDselect** to return a dataset 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 from **SDstart**
sds_index	IN:	Index of the dataset

Purpose Retrieves the *sds_id* for the given dataset.

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

Description To get an *sds_id* for the Nth dataset, use `N-1` as the index. `N` must be a number greater than or equal to `0` and less than the total number of datasets in the file. The total number of datasets in a file may be obtained from a call to **SDfileinfo**. The function **SDnametoindex** can be used to find the index of a dataset if its name is known.

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

FORTRAN

```
integer function sfselect(sd_id, sds_index)

integer sd_id, sds_index
```

SDsetaccesstype/sfsacct

intn SDsetaccesstype(int32 *sd_id*, int32 *access_type*)

sd_id	IN:	SD interface identifier returned from **SDstart**
access_type	IN:	I/O access mode of the dataset

Purpose Determines the I/O access mode to be used for the specified dataset.

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

Description The dataset must be an external element and the I/O access defined via this routine applies to the specified external file only.

The *access_type* parameter can be defined as DFACC_SERIAL for sequential I/O or DFACC_PARALLEL for parallel I/O. At present, HDF has only implemented the CM5 CMFS parallel I/O.

Though the access mode is set for dataset *sds_id*, it applies to the entire external file in which the dataset resides. If different access modes are defined for datasets within the same HDF external file, unpredictable effects will result. It is also the users' responsibility to make sure the access mode is appropriate for the files involved. For example, setting DFACC_PARALLEL on a dataset contained in a serial file may result in slower I/O rates and sometimes failures.

FORTRAN

```
integer function sfsacct(sd_id, access_type)

integer sd_id, access_type
```

SDsetattr/sfsnatt/sfscatt

intn SDsetattr(int32 [*file*, *sds*, *dim*]_*id*, char **attr_name*, int32 *data_type*, int32 *count*, VOIDP *values*)

[*file*, *sds*, *dim*]_*id*	IN:	Identifier of the object the attribute is to be attached to: an *file_id* for a file, an *sds_id* for an SDS or a *dim_id* for a dimension
attr_name	IN:	Name to be assigned to 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 storde in the attribute

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

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-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 data 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 datasets. For example, "file_creator_date" is a global attribute, whereas "units" is typically considered to be dataset-specific.

Valid values for *data_type* are prefaced by DFNT_. The following are valid symbolic names and their data types:

32-bit float	DFNT_FLOAT32	5
64-bit float	DFNT_FLOAT64	6
8-bit signed int	DFNT_INT8	20
8-bit unsigned int	DFNT_UINT8	21
16-bit signed int	DFNT_INT16	22
16-bit unsigned int	DFNT_UINT16	23
32-bit signed int	DFNT_INT32	24
32-bit unsigned int	DFNT_UINT32	25

```
8-bit signed character DFNT_CHAR   4
```

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.

Example Store a "valid_range" attribute for a dataset.

```
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 sfsnatt([file, sds, dim]_id, attr_name,
                         data_type, count, values)

character* (*) attr_name
integer [file, sds, dim]_id, data_type, count
<valid numeric data type> values(*)

integer function sfscatt([file, sds, dim]_id, attr_name,
                         data_type, count, values)

character* (*) attr_name, values
integer  [file, sds, dim]_id, data_type, count
```

SDsetblocksize

intn SDsetblocksize(int32 *sd_id*, int32 *block_size*)

sd_id	IN:	SD interface identifier returned from **SDstart**
block_size	IN:	Size of the block in bytes

Purpose Sets the block size used for storing data datasets with unlimited dimensions.

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

Description **SDsetblocksize** is used when creating new datasets only - it has no affect on pre-existing datasets. The *block_size* parameter should be set to a multiple of the "slice" size.

SDsetcal/sfscal

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

sds_id	IN:	Dataset identifier returned from **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 dataset. 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 `iy` stored in a dataset 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 dataset that is to be 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

intn SDsetchunk(int32 *sds_id*, HDF_CHUNK_DEF *cdef*, int32 *flags*)

sds_id	IN:	SD interface identifier returned from **SDstart**
cdef	IN:	Union containing information on how the chunks are to be defined
flags	IN:	Flags determining the behavior of the routine

Purpose Determines the chunk size and the compression method, if any, to be applied when partitioning the array into chunks.

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

Description **SDsetchunk** receives it's information on how to partition the array into chunks, as well as compression information, from a `HDF_CHUNK_DEF` union passed in as its second argument. This union structure is defined in the HDF library as follows:

```
typedef union hdf_chunk_def_u {
    int32 chunk_lengths[MAX_VAR_DIMS];
    struct {
        int32 chunk_lengths[MAX_VAR_DIMS];
        int32 comp_type;
        comp_info cinfo;
    } comp;
} HDF_CHUNK_DEF
```

The `flags` parameter can either be set to `HDF_CHUNK` if the SDS is to be uncompressed, or to the bitwise-OR'ed values of `HDF_CHUNK` and `HDF_COMP` (`HDF_CHUNK | HDF_COMP`) if a compression method is to be applied to the array while it is being partitioned into chunks.

SDsetchunkcache

intn SDsetchunkcache(int32 *sds_id*, int32 *maxcache*, int32 *flags*)

sds_id	IN:	SD interface identifier returned from **SDstart**
maxcache	IN:	Maximum number of chunks in the cache
flags	IN:	Flags determining the behavior of the routine

Purpose Determines the size of the chunk cache.

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

Description By default, when a generic SDS is promoted to be a chunked SDS, the `maxcache` parameter is set to the number of chunks along the last dimension and a cache for the chunks is created.

If the chunk cache is full and the value of the `maxcache` parameter is larger than the currently allowed maximum number of cached chunks, then the maximum number of cached chunks is reset to the value of `maxcache`. If the chunk cache is not full, then the size of the chunk cache is reset to the value of `maxcache` only if it is greater than current number of chunks in the cache.

Never set the value of `maxcache` to be less than the number of chunks along the last 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.

Currently the only allowed value of the `flags` parameter is `0`, which designates default operation. In the near future, the value `HDF_CACHEALL` will be supported to be used to specifiy that the whole SDS object is to be cached.

SDsetcompress

intn SDsetcompress(int32 *sd_id*, char **comp_type*, comp_info **cinfo*)

sd_id	IN:	SD interface identifier returned from **SDstart**
comp_type	IN:	Compression method
cinfo	IN:	Pointer to compression information structure

Purpose Sets the compression method for the specified dataset.

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

Description The **SDsetcompress** routine 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 `comp_type` parameter is the compression type definition and is set to `COMP_CODE_RLE` for run-length encoding, `COMP_CODE_DEFLATE` for Gnu ZIP (or GZIP) compression, `COMP_CODE_SKPHUFF` for skipping Huffman or `COMP_CODE_NONE` for no compression. The `c_info` parameter is a union structure of type `tag_comp_info`.

SDsetcompress compresses the dataset data at the time it is called, not during the next call to **SDwritedata**.

SDsetdatastrs/sfsdtstr

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

sds_id	IN:	Dataset identifier 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 dataset.

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

Description This routine can be used to set the label, unit, format, and coordinate system strings for a given dataset. `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 sds_id
character* (*) label, unit, format, coordsys
```

SDsetdimname/sfsdmname

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

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

Purpose Assigns a name to the specified dimension.

Return value Returns `SUCCEED` (or `0`) if successful and `FAIL` (or `-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 default names generated by the HDF library.

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 from **SDgetdimid**
count	IN:	Total number of values along the specified dimension
data_type	IN:	Data type of the values along the specified dimension
data	IN:	Value of each increment along the specified dimension

Purpose Stores the values of the specified dimension.

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

Description Users should note that it is possible to store values for a dimension without naming it. Even though its redundant, the *count* argument has been included for backward compatibility. Note that there is now a *data_type* parameter so the dimension scales are no longer required to be the same data type as the dataset.

FORTRAN

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

integer dim_id, count, data_type
character* (*) data
```

SDsetdimstrs/sfsdmstr

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

dim_id	IN:	Dimension identifier 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` (or `0`) if successful and `FAIL` (or `-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 dim_id
character* (*) label, unit, format
```

SDsetdimval_comp/sfsdmvc

intn SDsetdimval_comp(int32 *dim_id*, intn *comp_mode*)

dim_id	IN:	Dimension identifier returned from **SDgetdimid**
comp_mode	IN:	Compatibility mode to be set

Purpose Determines whether the specified dimension *will have* the old and new representations or the new representation only.

Return value Returns either SUCCEED (or 0) on successful completion, or FAIL (or -1) otherwise.

Description The `comp_mode` parameter is set to either `SD_DIMVAL_BW_COMP`, which specifies compatible mode and that the old and new dimension representations will be written to file , or `SD_DIMVAL_BW_INCOMP`, which specifies backward incompatible mode and that only the new dimension representation will be written to file.

Unlimited dimensions are always backward-compatable, therefore **SDsetdimval_comp** takes no action on unlimited dimensions.

As of HDF4.1r1, the default mode is backward-incompatable. Subsequent calls to **SDsetdimval_comp** will override the settings established in previous calls to the routine.‘

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:	Dataset identifier 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` (or `0`) if successful and `FAIL` (or `-1`) otherwise.

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

Data can only be moved once for any given dataset and it is the user's responsibility to make sure the external data file is kept with the "original" file. The *offset* parameter 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 post-version 3.2 files (i.e. calling on a netCDF file that was opened with the multi-file interface will fail).

If the SDS specified by *sds_id* already exists, its data will be moved to the external file and the connection between the tag and the data in the primary file will be broken, effectively making that data inaccessible to the interface routines. However, the data itself must be explicitly deleted by the `hdfpack` command-line utility, as **SDsetexternalfile** doesn't do this.

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 *file_id*, intn *fill_mode*)

file_id	IN:	Dataset identifier returned from **SDselect**
fill_mode	IN:	Fill mode

Purpose Sets the fill mode for the specified file.

Return value Returns the previous fill mode if successful and `FAIL` (or `-1`) otherwise.

Description **SDsetfillmode** sets the fill mode to be applied to all SDSs contained in the specified file.

When the specified file is first opened, or when the file is closed then re-opened, its fill mode will be the default SD_FILL mode, which indicates that fill values will be written when the SDS is created.

Possible values for `fill_mode` are: SD_FILL (or `0`) or SD_NOFILL (or 256), defined in "hdf.inc". SD_NOFILL indicates that fill values will not be written.

When an SDS without unlimited dimensions is created, by default the first **SDwritedata** call will fill the entire dataset with the user-defined fill value, or the default fill value if there is no user-defined fill value, during the call to **SDsetfillvalue**. In SDSs with an unlimited dimension defined as the slowest-changing 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 hasn't been specified.

If it is certain that all dataset values will be written before any read operation takes place, there is no need to write the fill values. Simply call **SDsetfillmode** with a `fill_mode` value of SD_NOFILL which will eliminate all fill value write operations to the dataset. For large datasets, this can improve the speed by almost 50%.

FORTRAN

```
integer function sfsflmd(file_id, fill_mode)

integer file_id, fill_mode
```

SDsetfillvalue/sfsfill

intn SDsetfillvalue(int32 *sds_id*, VOIDP *fill_value*)

sds_id	IN:	Dataset identifier returned from **SDselect**
fill_value	IN:	Fill value

Purpose Sets the fill value for the specified dataset.

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

Description The fill value is assumed to have the same data type as the dataset.

FORTRAN

```
integer function sfsfill(sds_id, fill_value)

integer sds_id, fill_value
```

SDsetnbitdataset/sfsnbit

intn SDsetnbitdataset(int32 *sds_id*, intn *start_bit*, intn *bit_len*, intn *sign_ext*, intn *fill_one*)

sds_id	IN:	Dataset identifier returned from **SDselect**
start_bit	IN:	Leftmost bit of the field to bewritten
bit_len	IN:	Length of the bit field to bewritten
sign_ext	IN:	Sign extend specifier
fill_one	IN:	Background bit specifier

Purpose Specifies a non-standard bit length for SDS data.

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

Description **SDnbitdataset** allows the HDF user to specify that a particular SDS array contains data of a non-standard length. Any length between 1 and 32 bits can be specified. After **SDnbitdataset** has been called for the SDS array, any read or write operations will involve a conversion between the new data length of the SDS 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 preceeding paragraph a `start_bit` parameter set to `4` would correspond to the 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 preceeding paragraph would result in the bit field `1110` being written to the dataset. 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 dataset 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 is set to either `TRUE` or `FALSE` - specify `TRUE` to sign-extend

The `fill_one` specifies whether to fill the "background" bits with the value `1` or `0`. This parameter is also set to either `TRUE` or `FALSE`.

The "background" bits of a variable-length dataset 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 dataset 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:	Dataset identifier returned from **SDselect**
max	IN:	Highest value in the range
min	IN:	Lowest value in the range

Purpose Sets the "valid" maximum and minimum values for the given dataset.

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

Description It is assumed that the data type for the maximum and minimum values are the same as the data 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-77 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 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 SDS access mode in effect during the current session

Purpose Opens the HDF file and initializes the SD interface.

Return value Returns an *sd_id* if successful and `FAIL` (or `-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* parameter is one of the following:

`DFACC_READ` - Open existing file for read-only access. If the file doesn't exist, specifying this mode will result in an error condition.

`DFACC_WRITE` - Open existing file for read and write access. If the file doesn't exist, specifying this mode will result in an error condition.

`DFACC_CREATE` - Create a new file with read and write access. If the file does exist, the contents of this file will be deleted before any new writes occur (the file contents will be replaced).

The file can be any one of the following:an XDR-based netCDF file, "old-style" DFSD file or a "new-style" SD file

If *access_mode* is set to `DFACC_CREATE` "new-style" SD files will be created. If *access_mode* is set to `DFACC_RDONLY`, the specified file will not be created if it doesn't exist.

The type of identifier returned by **SDstart** is currently not the same as the identifier 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 access_mode
```

SDwritechunk

intn SDwritechunk(int32 *sds_id*, int32 **origin*, VOIDP *datap*)

sds_id	IN:	SD interface identifier returned from **SDstart**
origin	IN:	Origin of the chunk to be written
datap	IN:	Buffer for the chunk data to be written

Purpose Writes data to a chunked scientific dataset.

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

Description **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 SDS.

Also, **SDwritechunk** is written specifically for chunked SDSs and doesn't have the overhead of the additional functionality supported by the **SDwritedata** routine - therefore, it is much faster than **SDwritedata**.

SDwritechunk will return `FAIL` when an attempt is made to use it to write to a non-chunked SDS.

SDwritedata/sfwdata/sfwcdata

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

sds_id	IN:	Dataset identifier 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 be written along each dimension
data	IN:	Values to be written to the dataset

Purpose Writes a hyperslab of data for a dataset.

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

Description The array *start* specifies 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 *stride* array 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` in the C interface or *stride* = `1` in either interface specifies the contiguous reading of data. If the *stride* values are set to `0`, **SDreaddata** returns `FAIL` (or `-1`). No matter what stride value is provided, data is always placed contiguously in buffer.

When writing data to a "chunked" SDS using **SDwritedata**, consideration should be given to be issues presented in the section on chunking in Chapter 3 of the HDF User's Manual, tited *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, data)

integer sds_id
integer start(*), stride(*), edge(*)
<valid numeric data type> data

integer function sfwcdata(sds_id, start, stride, edge, data)

integer sds_id
integer start(*), stride(*), edge(*)
character* (*) data
```