HDF5 Reference Manual

Release 1.8.3

May 2009

The HDF Group
http://wwww.hdfgroup.org/

(Printed: June 2008)

Table of Contents

* Links to descriptions of these tools appear on the HDF5 Tools page:
`http://hdfgroup.org/HDF5/doc/RM/Tools.html`

HDF5: API Specification Reference Manual

The HDF5 library provides several interfaces, each of which provides the tools required to meet specific aspects of the HDF5 data-handling requirements.

Notes regarding Fortran90 and C++ APIs appear on the next page.

Main HDF5 Library, or Low-level APIs

The main HDF5 Library includes all of the low-level APIs, providing user applications with fine-grain control of HDF5 functionality.

Library Functions	The general-purpose **H5** functions.
Attribute Interface	The **H5A** API for attributes.
Dataset Interface	The **H5D** API for manipulating scientific datasets.
Error Interface	The **H5E** API for error handling.
File Interface	The **H5F** API for accessing HDF5 files.
Group Interface	The **H5G** API for creating physical groups of objects on disk.
Identifier Interface	The **H5I** API for working with object identifiers.
Link Interface	The **H5L** API for working with links.
Object Interface	The **H5O** API for manipulating objects and reference counts.
Property List Interface	The **H5P** API for manipulating object property lists.
Reference Interface	The **H5R** API for references.
Dataspace Interface	The **H5S** API for defining dataset dataspace.
Datatype Interface	The **H5T** API for defining dataset element information.
Filters and Compression Interface	The **H5Z** API for inline data filters and data compression.
Tools	Interactive tools for the examination of existing HDF5 files.
Predefined Datatypes	Predefined datatypes in HDF5.

The Fortran90 and C++ APIs to HDF5

The HDF5 Library distribution includes FORTRAN90 and C++ APIs, which are described in the following documents.

Fortran90 API

Fortran90 APIs in the *HDF5 Reference Manual*: The *HDF5 Reference Manual* includes descriptions of the HDF5 Fortran90 APIs. Fortran subroutines exist in the H5, H5A, H5D, H5E, H5F, H5G, H5I, H5P, H5R, H5S, H5T, and H5Z interfaces and are described on those pages. In general, each Fortran subroutine performs exactly the same task as the corresponding C function, with which it is described.

HDF5 Fortran90 Flags, Datatypes and User’s Notes lists the flags employed in the Fortran90 interface, contains a pointer to the HDF5 Fortran90 datatypes, and includes the document *HDF5 Fortran90 User's Notes*.

HDF5 Fortran90 User’s Notes provides important information for users regarding the Fortran90 source code and the Fortran90 API.

C++ API

HDF5 C++ Reference Manual provides a complete reference for the HDF5 C++ interface.

H5: General Library Functions

These functions serve general-purpose needs of the HDF5 library and it users.

The C Interfaces:

- H5open
- H5close
- H5get_libversion
- H5check_version
- H5set_free_list_limits
- H5garbage_collect
- H5dont_atexit

Alphabetical Listing

- H5check_version
- H5close
- H5dont_atexit
- H5garbage_collect
- H5get_libversion
- H5open
- H5set_free_list_limits

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5open_f
- h5close_f
- h5get_libversion_f
- h5check_version_f
- h5set_free_list_limits_f
- h5garbage_collect_f
- h5dont_atexit_f

Name: H5check_version

Signature:

herr_t `H5check_version`(*unsigned* `majnum`, *unsigned* `minnum`, *unsigned* `relnum`)

Purpose:

Verifies that library versions are consistent.

Description:

`H5check_version` verifies that the arguments provided with the function call match the version numbers compiled into the library.

`H5check_version` serves two slightly differing purposes.

First, the function is intended to be called by the user to verify that the version of the header files compiled into an application matches the version of the HDF5 library being used. One may look at the `H5check` definition in the file `H5public.h` as an example.

Due to the risks of data corruption or segmentation faults, `H5check_version` causes the application to abort if the version numbers do not match. The abort is achieved by means of a call to the standard C function `abort()`.

Note that `H5check_version` verifies only the major and minor version numbers and the release number; it does not verify the sub-release value as that should be an empty string for any official release. This means that any two incompatible library versions must have different {major,minor,release} numbers. (Notice the reverse is not necessarily true.)

Secondarily, `H5check_version` verifies that the library version identifiers `H5_VERS_MAJOR`, `H5_VERS_MINOR`, `H5_VERS_RELEASE`, `H5_VERS_SUBRELEASE`, and `H5_VERS_INFO` are consistent. This is designed to catch source code inconsistencies, but does not generate the fatal error as in the first stage because this inconsistency does not cause errors in the data files. If this check reveals inconsistencies, the library issues a warning but the function does not fail.

Parameters:

unsigned `majnum`	IN: The major version of the library.
unsigned `minnum`	IN: The minor version of the library.
unsigned `relnum`	IN: The release number of the library.

Returns:

Returns a non-negative value if successful. Upon failure, this function causes the application to abort.

Fortran90 Interface: h5check_version_f

```
SUBROUTINE h5check_version_f(majnum, minnum, relnum, hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(IN)  :: majnum      ! The major version of the library
  INTEGER, INTENT(IN)  :: minnum      ! The minor version of the library
  INTEGER, INTENT(IN)  :: relnum      ! The release number
  INTEGER, INTENT(OUT) :: hdferr      ! Error code

END SUBROUTINE h5check_version_f
```

History:

Release	**Fortran90**
1.4.5	Function introduced in this release.

Name: H5close
Signature:

herr_t `H5close`(*void*)

Purpose:

Flushes all data to disk, closes file identifiers, and cleans up memory.

Description:

`H5close` flushes all data to disk, closes all file identifiers, and cleans up all memory used by the library. This function is generally called when the application calls `exit()`, but may be called earlier in event of an emergency shutdown or out of desire to free all resources used by the HDF5 library.

`h5close_f` and `h5open_f` are required calls in Fortran90 applications.

Parameters:

None.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5close_f

```
SUBROUTINE h5close_f(hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(OUT) :: hdferr      ! Error code

END SUBROUTINE h5close_f
```

Name: H5dont_atexit
Signature:

herr_t `H5dont_atexit`(*void*)

Purpose:

Instructs library not to install `atexit` cleanup routine.

Description:

`H5dont_atexit` indicates to the library that an `atexit()` cleanup routine should not be installed. The major purpose for this is in situations where the library is dynamically linked into an application and is un-linked from the application before `exit()` gets called. In those situations, a routine installed with `atexit()` would jump to a routine which was no longer in memory, causing errors.

In order to be effective, this routine *must* be called before any other HDF function calls, and must be called each time the library is loaded/linked into the application (the first time and after it's been un-loaded).

Parameters:
None.
Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5dont_atexit_f

```
SUBROUTINE h5dont_atexit_f(hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(OUT) :: hdferr      ! Error code

END SUBROUTINE h5dont_atexit_f
```

History:

Release	**Fortran90**
1.4.5	Function introduced in this release.

Name: H5garbage_collect

Signature:

herr_t `H5garbage_collect`(*void*)

Purpose:

Garbage collects on all free-lists of all types.

Description:

`H5garbage_collect` walks through all the garbage collection routines of the library, freeing any unused memory.

It is not required that `H5garbage_collect` be called at any particular time; it is only necessary in certain situations where the application has performed actions that cause the library to allocate many objects. The application should call `H5garbage_collect` if it eventually releases those objects and wants to reduce the memory used by the library from the peak usage required.

The library automatically garbage collects all the free lists when the application ends.

Parameters:

None.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5garbage_collect_f

```
SUBROUTINE h5garbage_collect_f(hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(OUT) :: hdferr      ! Error code

END SUBROUTINE h5garbage_collect_f
```

History:

Release	**Fortran90**
1.4.5	Function introduced in this release.

Name: H5get_libversion

Signature:

herr_t `H5get_libversion`(*unsigned* *`majnum`, *unsigned* *`minnum`, *unsigned* *`relnum`)

Purpose:

Returns the HDF library release number.

Description:

`H5get_libversion` retrieves the major, minor, and release numbers of the version of the HDF library which is linked to the application.

Parameters:

unsigned *`majnum`	OUT: The major version of the library.
unsigned *`minnum`	OUT: The minor version of the library.
unsigned *`relnum`	OUT: The release number of the library.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5get_libversion_f

```
SUBROUTINE h5get_libversion_f(hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(OUT) :: majnum      ! The major version of the library
  INTEGER, INTENT(OUT) :: minnum      ! The minor version of the library
  INTEGER, INTENT(OUT) :: relnum      ! The release number
  INTEGER, INTENT(OUT) :: hdferr      ! Error code

END SUBROUTINE h5get_libversion_f
```

History:

Release	**Fortran90**
1.4.5	Function introduced in this release.

Name: H5open

Signature:

herr_t `H5open`(*void*)

Purpose:

Initializes the HDF5 library.

Description:

`H5open` initialize the library.

When the HDF5 Library is employed in a C application, this function is normally called automatically, but if you find that an HDF5 library function is failing inexplicably, try calling this function first. If you wish to elimnate this possibility, it is safe to routinely call `H5open` before an application starts working with the library as there are no damaging side-effects in calling it more than once.

When the HDF5 Library is employed in a Fortran90 application, `h5open_f` initializes global variables (e.g. predefined types) and performs other tasks required to initialize the library. `h5open_f` and `h5close_f` are therefore required calls in Fortran90 applications.

Parameters:

None.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5open_f

```
SUBROUTINE h5open_f(hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(OUT) :: hdferr      ! Error code

END SUBROUTINE h5open_f
```

Name: H5set_free_list_limits

Signature:

herr_t `H5set_free_list_limits`(*int* `reg_global_lim`, *int* `reg_list_lim`, *int* `arr_global_lim`, *int* `arr_list_lim`, *int* `blk_global_lim`, *int* `blk_list_lim`)

Purpose:

Sets free-list size limits.

Description:

`H5set_free_list_limits` sets size limits on all types of free lists. The HDF5 library uses free lists internally to manage memory. The types of free lists used are as follows:

- ◊ Regular free lists manage memory for single internal data structures.
- ◊ Array free lists manage memory for arrays of internal data structures.
- ◊ Block free lists manage memory for arbitrarily-sized blocks of bytes.
- ◊ Factory free lists manage memory for fixed-size blocks of bytes.

The parameters specify global and per-list limits; for example, `reg_global_limit` and `reg_list_limit` limit the accumulated size of all regular free lists and the size of each individual regular free list, respectively. Therefore, if an application sets a 1Mb limit on each of the global lists, up to 4Mb of total storage might be allocated, 1Mb for each of the regular, array, block, and factory type lists.

The settings specified for block free lists are duplicated for factory free lists. Therefore, increasing the global limit on block free lists by *x* bytes will increase the potential free list memory usage by 2*x* bytes.

Using a value of `-1` for a limit means that no limit is set for the specified type of free list.

Parameters:

int `reg_global_lim`	IN: The cumulative limit on all regular free list memory used
int `reg_list_lim`	IN: The limit on memory used for each regular free list
int `arr_global_lim`	IN: The cumulative limit on all array free list memory used
int `arr_list_lim`	IN: The limit on memory used for each array free list
int `blk_global_lim`	IN: The cumulative limit on each of block free list memory and factory free list memory used
int `blk_list_lim`	IN: The limit on memory used for each block or factory free list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.
1.8.3	Function changed in this release to set factory free list memory limits.

H5A: Attribute Interface

Attribute API Functions

These functions create and manipulate attributes and information about attributes.
In the following lists, italic type indicates a configurable macro.

The C Interfaces:

- *H5Acreate*
- H5Acreate1 *
- H5Acreate2
- H5Acreate_by_name
- H5Aopen
- H5Aopen_by_name
- H5Aopen_name *
- H5Aopen_by_idx
- H5Aopen_idx *
- H5Aexists
- H5Aexists_by_name

- H5Arename
- H5Arename_by_name
- H5Awrite
- H5Aread
- H5Aclose
- *H5Aiterate*
- H5Aiterate1 *
- H5Aiterate2
- H5Aiterate_by_name
- H5Adelete
- H5Adelete_by_name
- H5Adelete_by_idx

- H5Aget_info
- H5Aget_info_by_name
- H5Aget_info_by_idx
- H5Aget_num_attrs *
- H5Aget_name
- H5Aget_create_plist
- H5Aget_space
- H5Aget_type
- H5Aget_storage_size
- H5Aget_name_by_idx

** Use of these functions is deprecated in Release 1.8.0.*

Alphabetical Listing

- H5Aclose
- *H5Acreate*
- H5Acreate1 *
- H5Acreate2
- H5Acreate_by_name
- H5Adelete
- H5Adelete_by_name
- H5Adelete_by_idx
- H5Aexists
- H5Aexists_by_name
- H5Aget_create_plist

- H5Aget_info
- H5Aget_info_by_idx
- H5Aget_info_by_name
- H5Aget_name
- H5Aget_name_by_idx
- H5Aget_num_attrs *
- H5Aget_space
- H5Aget_storage_size
- H5Aget_type
- *H5Aiterate*
- H5Aiterate1 *

- H5Aiterate2
- H5Aiterate_by_name
- H5Aopen
- H5Aopen_by_idx
- H5Aopen_by_name
- H5Aopen_idx *
- H5Aopen_name *
- H5Aread
- H5Arename
- H5Arename_by_name
- H5Awrite

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5aclose_f
- h5acreate_f
- h5acreate_by_name_f
- h5adelete_f
- h5adelete_by_name_f
- h5adelete_by_idx_f
- H5Aexists_f
- H5Aexists_by_name_f
- h5aget_create_plist_f
- h5aget_info_f
- h5aget_info_by_idx_f
- h5aget_info_by_name_f
- h5aget_name_f
- h5aget_name_by_idx_f
- h5aget_num_attrs_f *
- h5aget_space_f
- h5aget_storage_size_f
- h5aget_type_f
- h5aopen_f
- h5aopen_by_idx_f
- h5aopen_by_name_f
- h5aopen_idx_f *
- h5aopen_name_f *
- h5aread_f
- h5arename_f
- h5arename_by_name_f
- h5awrite_f

** Use of these functions is deprecated in Release 1.8.0.*

The Attribute interface, H5A, is primarily designed to easily allow small datasets to be attached to primary datasets as metadata information. Additional goals for the H5A interface include keeping storage requirement for each attribute to a minimum and easily sharing attributes among datasets.

Because attributes are intended to be small objects, large datasets intended as additional information for a primary dataset should be stored as supplemental datasets in a group with the primary dataset. Attributes can then be attached to the group containing everything to indicate a particular type of dataset with supplemental datasets is located in the group. How small is "small" is not defined by the library and is up to the user’s interpretation.

See *Attributes* in the *HDF5 User's Guide* for further information.

Name: H5Aclose
Signature:

herr_t `H5Aclose`(*hid_t* `attr_id`)

Purpose:

Closes the specified attribute.

Description:

`H5Aclose` terminates access to the attribute specified by `attr_id` by releasing the identifier.

Further use of a released attribute identifier is illegal; a function using such an identifier will fail.

Parameters:

hid_t `attr_id`	IN: Attribute to release access to.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5aclose_f

```
SUBROUTINE h5aclose_f(attr_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(OUT) :: attr_id  ! Attribute identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code:
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5aclose_f
```

Name: H5Acreate

Signature:

hid_t `H5Acreate`(*hid_t* `loc_id`, *const char* *`attr_name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `acpl_id`) [1]

hid_t `H5Acreate`(*hid_t* `loc_id`, *const char* *`attr_name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `acpl_id`, *hid_t* `aapl_id`) [2]

Purpose:

Creates an attribute attached to a specified object.

Description:

`H5Acreate` is a macro that is mapped to either `H5Acreate1` or `H5Acreate2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

- ◊ The `H5Acreate` macro will be mapped to `H5Acreate1` and will use the `H5Acreate1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.
- ◊ The `H5Acreate` macro mapped to `H5Acreate2` and will use the `H5Acreate2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Acreate` is mapped to the most recent version of the function, currently `H5Acreate2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Acreate` will be mapped to `H5Acreate1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Acreate` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Acreate2`
Enable deprecated symbols	`H5Acreate2`
Disable deprecated symbols	`H5Acreate2`
Emulate Release 1.6 interface	`H5Acreate1`
<u>Function-level macros</u>	
`H5Acreate_vers = 2`	`H5Acreate2`
`H5Acreate_vers = 1`	`H5Acreate1`

Interface history: Signature [1] above is the original `H5Acreate` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Acreate1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Acreate2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface: h5acreate_f

```
SUBROUTINE h5acreate_f(loc_id, name, type_id, space_id, attr_id, hdferr, &
                       acpl_id, aapl_id )
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id    ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: name    ! Attribute name
  INTEGER(HID_T), INTENT(IN) :: type_id   ! Attribute datatype identifier
  INTEGER(HID_T), INTENT(IN) :: space_id  ! Attribute dataspace identifier
  INTEGER(HID_T), INTENT(OUT) :: attr_id  ! Attribute identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code:
                                          ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: acpl_id
                                          ! Attribute creation property
                                          ! list identifier
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: aapl_id
                                          ! Attribute access property
                                          ! list identifier
END SUBROUTINE h5acreate_f
```

History:

Release	**C**
1.8.0	The function `H5Acreate` renamed to `H5Acreate1` and deprecated in this release. The macro `H5Acreate` and the functions `H5Acreate2` and `H5Acreate_by_name` introduced in this release.

Name: H5Acreate1

Signature:

hid_t `H5Acreate1`(*hid_t* `loc_id`, *const char* *`attr_name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `acpl_id`)

Purpose:

Creates a dataset as an attribute of another group, dataset, or named datatype.

Notice:

This function is deprecated in favor of the function `H5Acreate2`.

Description:

`H5Acreate1` creates the attribute `attr_name` attached to the object specified with `loc_id`. `loc_id` can be a group, dataset, or named datatype identifier.

The attribute name specified in `attr_name` must be unique. Attempting to create an attribute with the same name as an already existing attribute will fail, leaving the pre-existing attribute in place. To overwrite an existing attribute with a new attribute of the same name, first call `H5Adelete` then recreate the attribute with `H5Acreate1`.

The datatype and dataspace identifiers of the attribute, `type_id` and `space_id`, respectively, are created with the H5T and H5S interfaces, respectively.

Currently only simple dataspaces are allowed for attribute dataspaces.

The attribute creation property list, `acpl_id`, is currently unused; it may be used in the future for optional attribute properties. At this time, `H5P_DEFAULT` is the only accepted value.

The attribute identifier returned from this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Object (dataset, group, or named datatype) to be attached to.
const char *`attr_name`	IN: Name of attribute to create.
hid_t `type_id`	IN: Identifier of datatype for attribute.
hid_t `space_id`	IN: Identifier of dataspace for attribute.
hid_t `acpl_id`	IN: Identifier of creation property list. *(Currently unused; the only accepted value is `H5P_DEFAULT`.)*

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Acreate`.

History:

Release	**C**
1.8.0	The function `H5Acreate` renamed to `H5Acreate1` and deprecated in this release.

Name: H5Acreate2

Signature:

hid_t `H5Acreate2`(*hid_t* `loc_id`, *const char* *`attr_name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `acpl_id`, *hid_t* `aapl_id`,)

Purpose:

Creates an attribute attached to a specified object.

Description:

`H5Acreate2` creates an attribute, `attr_name`, which is attached to the object specified by the identifier `loc_id`.

The attribute name, `attr_name`, must be unique for the object.

The attribute is created with the specified datatype and dataspace, `type_id` and `space_id`, which are created with the H5T and H5S interfaces respectively.

The attribute creation and access property lists are currently unused, but will be used in the future for optional attribute creation and access properties. These property lists should currently be `H5P_DEFAULT`.

The attribute identifier returned by this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Location or object identifier; may be dataset or group
const char *`attr_name`	IN: Attribute name
hid_t `type_id`	IN: Attribute datatype identifier
hid_t `space_id`	IN: Attribute dataspace identifier
hid_t `acpl_id`	IN: Attribute creation property list identifier *(Currently not used.)*
hid_t `aapl_id`	IN: Attribute access property list identifier *(Currently not used.)*

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Acreate`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Acreate_by_name

Signature:

hid_t `H5Acreate_by_name`(*hid_t* `loc_id`, *const char* *`obj_name`, *const char* *`attr_name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `acpl_id`, *hid_t* `aapl_id`, *hid_t* `lapl_id`)

Purpose:

Creates an attribute attached to a specified object.

Description:

`H5Acreate_by_name` creates an attribute, `attr_name`, which is attached to the object specified by `loc_id` and `obj_name`.

`loc_id` is a location identifier; `obj_name` is the object name relative to `loc_id`. If `loc_id` fully specifies the object to which the attribute is to be attached, `obj_name` should be `'.'` (a dot).

The attribute name, `attr_name`, must be unique for the object.

The attribute is created with the specified datatype and dataspace, `type_id` and `space_id`, which are created with the H5T and H5S interfaces respectively.

The attribute creation and access property lists are currently unused, but will be used in the future for optional attribute creation and access properties. These property lists should currently be `H5P_DEFAULT`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

The attribute identifier returned by this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Location or object identifier; may be dataset or group
const char *`obj_name`	IN: Name, relative to `loc_id`, of object that attribute is to be attached to
const char *`attr_name`	IN: Attribute name
hid_t `type_id`	IN: Attribute datatype identifier
hid_t `space_id`	IN: Attribute dataspace identifier
hid_t `acpl_id`	IN: Attribute creation property list identifier *(Currently not used.)*
hid_t `aapl_id`	IN: Attribute access property list identifier *(Currently not used.)*
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5acreate_by_name_f

```
SUBROUTINE h5acreate_by_name_f(loc_id, obj_name, attr_name, type_id, space_id, &
     attr, hdferr, acpl_id, aapl_id, lapl_id)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id      ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: obj_name  ! Name of object to which
                                            ! attribute is attached
  CHARACTER(LEN=*), INTENT(IN) :: attr_name ! Attribute name
  INTEGER(HID_T), INTENT(IN) :: type_id     ! Attribute datatype identifier
```

```
    INTEGER(HID_T), INTENT(IN) :: space_id    ! Attribute dataspace identifier
    INTEGER(HID_T), INTENT(OUT) :: attr       ! An attribute identifier
    INTEGER, INTENT(OUT) :: hdferr            ! Error code:
                                              ! 0 on success and -1 on failure
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: acpl_id
                                              ! Attribute creation property list
                                              ! identifier (Currently not used.)
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: aapl_id
                                              ! Attribute access property list
                                              ! identifier (Currently not used.)
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                              ! Link access property list
  END SUBROUTINE h5acreate_by_name_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Adelete

Signature:

herr_t `H5Adelete`(*hid_t* `loc_id`, *const char* *`attr_name`)

Purpose:

Deletes an attribute from a specified location.

Description:

`H5Adelete` removes the attribute specified by its name, `attr_name`, from a dataset, group, or named datatype. This function should not be used when attribute identifiers are open on `loc_id` as it may cause the internal indexes of the attributes to change and future writes to the open attributes to produce incorrect results.

Parameters:

hid_t `loc_id`	IN: Identifier of the dataset, group, or named datatype to have the attribute deleted from.
const char *`attr_name`	IN: Name of the attribute to delete.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5adelete_f

```
SUBROUTINE h5adelete_f(obj_id, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id    ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: name    ! Attribute name
  INTEGER, INTENT(OUT) :: hdferr          ! Error code:
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5adelete_f
```

History:

Release	**C**

Name: H5Adelete_by_name

Signature:

herr_t `H5Adelete_by_name`(*hid_t* `loc_id`, *const char* *`obj_name`, *const char* *`attr_name`, *hid_t* `lapl_id`)

Purpose:

Removes an attribute from a specified location.

Description:

`H5Adelete_by_name` removes the attribute `attr_name` from an object specified by location and name, `loc_id` and `obj_name`, respectively.

If `loc_id` fully specifies the object from which the attribute is to be removed, `obj_name` should be `'.'` (a dot).

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

Parameters:

hid_t `loc_id`	IN: Location or object identifier; may be dataset or group
const char *`obj_name`	IN: Name of object, relative to location, from which attribute is to be removed
const char *`attr_name`	IN: Name of attribute to delete
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5adelete_by_name_f

```
SUBROUTINE h5adelete_by_name_f(loc_id, obj_name, attr_name, hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! Identifer for object to which
                                         ! attribute is attached
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                                         ! Name of object, relative to location,
                                         ! from which attribute is to be removed
  CHARACTER(LEN=*), INTENT(IN) :: attr_name
                                         ! Name of attribute to delete
  INTEGER, INTENT(OUT) :: hdferr         ! Error code:
                                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                         ! Link access property list
END SUBROUTINE h5adelete_by_name_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Adelete_by_idx
Signature:

herr_t `H5Adelete_by_idx`(*hid_t* `loc_id`, *const char* *`obj_name`, *H5_index_t* `idx_type`, *H5_iter_order_t* `order`, *hsize_t* `n`, *hid_t* `lapl_id`)

Purpose:

Deletes an attribute from an object according to index order.

Description:

`H5Adelete_by_idx` removes an attribute, specified by its location in an index, from an object.

The object from which the attribute is to be removed is specified by a location identifier and name, `loc_id` and `obj_name`, respectively. If `loc_id` fully specifies the object from which the attribute is to be removed, `obj_name` should be `'.'` (a dot).

The attribute to be removed is specified by a position in an index, `n`. The type of index is specified by `idx_type` and may be `H5_INDEX_NAME`, for an alpha-numeric index by name, or `H5_INDEX_CRT_ORDER`, for an index by creation order. The order in which the index is to be traversed is specified by `order` and may be `H5_ITER_INC` (increment) for top-down iteration, `H5_ITER_DEC` (decrement) for bottom-up iteration, or `H5_ITER_NATIVE`, in which case HDF5 will iterate in the fastest-available order. For example, if `idx_type`, `order`, and `n` are set to `H5_INDEX_NAME`, `H5_ITER_INC`, and `5`, respectively, the fifth attribute by alpha-numeric order of attribute names will be removed.

For a discussion of `idx_type` and `order`, the valid values of those parameters, and the use of `n`, see the description of H5Aiterate2

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

Parameters:

hid_t `loc_id`	IN: Location or object identifier; may be dataset or group
const char *`obj_name`	IN: Name of object, relative to location, from which attribute is to be removed
H5_index_t `idx_type`	IN: Type of index
H5_iter_order_t `order`	IN: Order in which to iterate over index
hsize_t `n`	IN: Offset within index
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5adelete_by_idx_f

```
SUBROUTINE h5adelete_by_idx_f(loc_id, obj_name, idx_type, order, n,  hdferr, &
     lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T),INTENT(IN) :: loc_id
                             ! Identifer for object to which
                             ! attribute is attached
  CHARACTER(LEN=*),INTENT(IN) :: obj_name
                             ! Name of object, relative to location,
                             ! from which attribute is to be removed
  INTEGER, INTENT(IN) :: idx_type
                             ! Type of index; Possible values are:
                             !  H5_INDEX_UNKNOWN_F     - Unknown index type
```

```
                            !    H5_INDEX_NAME_F      - Index on names
                            !    H5_INDEX_CRT_ORDER_F - Index on creation order
                            !    H5_INDEX_N_F         - Number of indices defined
  INTEGER, INTENT(IN) :: order
                            ! Order in which to iterate over index:
                            !    H5_ITER_UNKNOWN_F  - Unknown order
                            !    H5_ITER_INC_F      - Increasing order
                            !    H5_ITER_DEC_F      - Decreasing order
                            !    H5_ITER_NATIVE_F   - No particular order,
                            !                         whatever is fastest
                            !    H5_ITER_N_F        - Number of iteration orders
  INTEGER(HSIZE_T), INTENT(IN) :: n
                            ! Offset within index
  INTEGER, INTENT(OUT) :: hdferr
                            ! Error code:
                            ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                            ! Link access property list
END SUBROUTINE h5adelete_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aexists

Signature:

htri_t `H5Aexists`(*hid_t* `obj_id`, *const char* *`attr_name`)

Purpose:

Determines whether an attribute with a given name exists on an object.

Description:

`H5Aexists` determines whether the attribute `attr_name` exists on the object specified by `obj_id`.

Parameters:

hid_t `obj_id`,	IN: Object identifier
const char *`attr_name`	IN: Attribute name

Returns:

When successful, returns a positive value, for `TRUE`, or `0` (zero), for `FALSE`.
Otherwise returns a negative value.

Fortran90 Interface: h5aexists_f

```
SUBROUTINE h5aexists_f(obj_id, attr_name, attr_exists, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id      ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: attr_name ! Attribute name
  LOGICAL, INTENT(OUT) :: attr_exists       ! .TRUE. if exists, .FALSE. otherwise
  INTEGER, INTENT(OUT) :: hdferr            ! Error code:
                                            ! 0 on success and -1 on failure
END SUBROUTINE
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aexists_by_name

Signature:

htri_t `H5Aexists_by_name`(*hid_t* `loc_id`, *const char* *`obj_name`, *const char* *`attr_name`, *hid_t* `lapl_id`)

Purpose:

Determines whether an attribute with a given name exists on an object.

Description:

`H5Aexists_by_name` determines whether the attribute `attr_name` exists on an object. That object is specified by its location and name, `loc_id` and `obj_name`, respectively.

`loc_id` specifies a location in the file containing the object. `obj_name` is the name of the object to which the attribute is attached and can be a relative name, relative to `loc_id`, or an absolute name, based in the root group of the file. If `loc_id` fully specifies the object, `obj_name` should be `'.'` (a dot).

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access `obj_name`. See “Link Access Properties” in the H5P APIs.

Parameters:

hid_t `loc_id`,	IN: Location identifier
const char *`obj_name`	IN: Object name Either relative to `loc_id`, absolute from the file’s root group, or `'.'` (a dot)
const char *`attr_name`	IN: Attribute name
hid_t `lapl_id`	IN: Link access property list identifier

Returns:

When successful, returns a positive value, for `TRUE`, or `0` (zero), for `FALSE`.
Otherwise returns a negative value.

Fortran90 Interface: h5aexists_by_name_f

```
SUBROUTINE h5aexists_by_name_f(loc_id, obj_name, attr_name, attr_exists, hdferr,&
     lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id  ! Location identifier
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                                        ! Object name either relative to loc_id,
                                        ! absolute from the
                                        ! file’s root group, or '.'
  CHARACTER(LEN=*), INTENT(IN) :: attr_name
                                        ! Attribute name
  LOGICAL, INTENT(OUT) :: attr_exists   ! .TRUE. if exists, .FALSE. otherwise
  INTEGER, INTENT(OUT) :: hdferr        ! Error code:
                                        ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                        ! Link access property list identifier
END SUBROUTINE h5aexists_by_name_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aget_create_plist

Signature:

hid_t `H5Aget_create_plist`(*hid_t* `attr_id`)

Purpose:

Gets an attribute creation property list identifier.

Description:

`H5Aget_create_plist` returns an identifier for the attribute creation property list associated with the attribute specified by `attr_id`.

The creation property list identifier should be released with `H5Pclose`.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute.

Returns:

Returns an identifier for the attribute's creation property list if successful. Otherwise returns a negative value.

Fortran90 Interface: h5aget_create_plist_f

```
SUBROUTINE h5aget_create_plist_f(attr_id, creation_prop_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id
                            ! Identifier of the attribute
  INTEGER(HID_T), INTENT(OUT) :: creation_prop_id
                            ! Identifier for the attribute's creation property
  INTEGER, INTENT(OUT) :: hdferr
                            ! Error code:
                            ! 0 on success and -1 on failure
END SUBROUTINE h5aget_create_plist_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aget_info

Signature:

herr_t `H5Aget_info`(*hid_t* `attr_id`, *H5A_info_t* *`ainfo`)

Purpose:

Retrieves attribute information, by attribute identifier.

Description:

`H5Aget_info` retrieves attribute information, locating the attribute with an attribute identifier, `attr_id`, which is the identifier returned by `H5Aopen` or `H5Aopen_by_idx`. The attribute information is returned in the `ainfo` struct.

The `ainfo` struct is defined as follows:

```
typedef struct {
    hbool_t            corder_valid;
    H5O_msg_crt_idx_t  corder;
    H5T_cset_t         cset;
    hsize_t            data_size;
} H5A_info_t;
```

`corder_valid` indicates whether the creation order data is valid for this attribute. Note that if creation order is not being tracked, no creation order data will be valid. Valid values are `TRUE` and `FALSE`.

`corder` is a positive integer containing the creation order of the attribute. This value is 0-based, so, for example, the third attribute created will have a `corder` value of `2`.

`cset` indicates the character set used for the attribute's name; valid values are defined in `H5Tpublic.h` and include the following:

`H5T_CSET_ASCII`	US ASCII
`H5T_CSET_UTF8`	UTF-8 Unicode encoding

This value is set with `H5Pset_char_encoding`.

`data_size` indicates the size, in the number of characters, of the attribute.

Parameters:

hid_t `attr_id`	IN: Attribute identifier
H5A_info_t *`ainfo`	OUT: Attribute information struct

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5aget_info_f

```
SUBROUTINE h5aget_info_f(attr_id, f_corder_valid, corder, cset, data_size,hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id  ! Attribute identifier
  LOGICAL, INTENT(OUT) :: f_corder_valid ! Indicates whether the creation order
                                         ! data is valid for this attribute
  INTEGER, INTENT(OUT) :: corder         ! Is a positive integer containing the
                                         ! creation order of the attribute
  INTEGER, INTENT(OUT) :: cset           ! Indicates the character set used for
                                         ! the ! attribute's name
  INTEGER(HSIZE_T), INTENT(OUT) :: data_size
                                         ! Indicates the size, in the number
```

```
                                                 ! of characters, of the attribute
          INTEGER, INTENT(OUT) :: hdferr         ! Error code:
                                                 ! 0 on success and -1 on failure
        END SUBROUTINE h5aget_info_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aget_info_by_idx

Signature:

herr_t `H5Aget_info_by_idx`(*hid_t* `loc_id`, *const char* *`obj_name`, *H5_index_t* `idx_type`, *H5_iter_order_t* `order`, *hsize_t* `n`, *H5A_info_t* *`ainfo`, *hid_t* `lapl_id`)

Purpose:

Retrieves attribute information, by attribute index position.

Description:

`H5Aget_info_by_idx` retrieves information for an attribute that is attached to an object, which is specified by its location and name, `loc_id` and `obj_name`, respectively. The attribute is located by its index position and the attribute information is returned in the `ainfo` struct.

If `loc_id` fully specifies the object to which the attribute is attached, `obj_name` should be `'.'` (a dot).

The attribute is located by means of an index type, an index traversal order, and a position in the index, `idx_type`, `order` and `n`, respectively. These parameters and their valid values are discussed in the description of `H5Aiterate2`.

The `ainfo` struct, which will contain the returned attribute information, is described in `H5Aget_info`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

Parameters:

hid_t `loc_id`	IN: Location of object to which attribute is attached
const char *`obj_name`	IN: Name of object to which attribute is attached, relative to location
H5_index_t `idx_type`	IN: Type of index
H5_iter_order_t `order`	IN: Index traversal order
hsize_t `n`	IN: Attribute’s position in index
H5A_info_t *`ainfo`	OUT: Struct containing returned attribute information
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5aget_info_by_idx_f

```
SUBROUTINE h5aget_info_by_idx_f(loc_id, obj_name, idx_type, order, n, &
     f_corder_valid, corder, cset, data_size, hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                              ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                              ! Name of object to which attribute is attached
  INTEGER, INTENT(IN) :: idx_type
                              ! Type of index; Possible values are:
                              !   H5_INDEX_UNKNOWN_F   - Unknown index type
                              !   H5_INDEX_NAME_F      - Index on names
                              !   H5_INDEX_CRT_ORDER_F - Index on creation order
                              !   H5_INDEX_N_F         - Number of indices defined
  INTEGER, INTENT(IN) :: order
                              ! Order in which to iterate over index:
                              !   H5_ITER_UNKNOWN_F  - Unknown order
                              !   H5_ITER_INC_F      - Increasing order
                              !   H5_ITER_DEC_F      - Decreasing order
                              !   H5_ITER_NATIVE_F   - No particular order,
                              !                        whatever is fastest
  INTEGER(HSIZE_T), INTENT(IN) :: n
                              ! Attribute’s position in index

  LOGICAL, INTENT(OUT) :: f_corder_valid
                              ! Indicates whether the creation order data is
                              ! valid for this attribute
  INTEGER, INTENT(OUT) :: corder
                              ! Is a positive integer containing the creation
                              !order of the attribute
  INTEGER, INTENT(OUT) :: cset
                              ! Indicates the character set used for the
                              ! attribute’s name
  INTEGER(HSIZE_T), INTENT(OUT) :: data_size
                              ! Indicates the size, in the number of characters,
                              ! of the attribute
  INTEGER, INTENT(OUT) :: hdferr
                              ! Error code:
                              ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                              ! Link access property list
END SUBROUTINE h5aget_info_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aget_info_by_name

Signature:

herr_t `H5Aget_info_by_name`(*hid_t* `loc_id`, *const char* *`obj_name`, *const char* *`attr_name`, *H5A_info_t* *`ainfo`, *hid_t* `lapl_id`)

Purpose:

Retrieves attribute information, by attribute name.

Description:

`H5Aget_info_by_name` retrieves information for an attribute, `attr_name`, that is attached to an object, specified by its location and name, `loc_id` and `obj_name`, respectively. The attribute information is returned in the `ainfo` struct.

If `loc_id` fully specifies the object to which the attribute is attached, `obj_name` should be `'.'` (a dot).

The `ainfo` struct is described in `H5Aget_info`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

Parameters:

hid_t `loc_id`	IN: Location of object to which attribute is attached
const char *`obj_name`	IN: Name of object to which attribute is attached, relative to location
const char *`attr_name`	IN: Attribute name
H5A_info_t *`ainfo`	OUT: Struct containing returned attribute information
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5aget_info_by_name_f

```
SUBROUTINE h5aget_info_by_name_f(loc_id, obj_name, attr_name, &
     f_corder_valid, corder, cset, data_size, hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id       ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: obj_name   ! Name of object to which attribute
                                             ! is attached
  CHARACTER(LEN=*), INTENT(IN) :: attr_name  ! Attribute name
  LOGICAL, INTENT(OUT) :: f_corder_valid     ! Indicates whether the creation
                                             ! order data is valid for this
                                             ! attribute
  INTEGER, INTENT(OUT) :: corder             ! Is a positive integer containing
                                             ! the creation order of the
                                             ! attribute
  INTEGER, INTENT(OUT) :: cset               ! Indicates the character set used
                                             ! for the attribute’s name
  INTEGER(HSIZE_T), INTENT(OUT) :: data_size ! Indicates the size, in the number
                                             ! of characters, of the attribute
  INTEGER, INTENT(OUT) :: hdferr             ! Error code:
                                             ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                             ! Link access property list
END SUBROUTINE h5aget_info_by_name_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aget_name

Signature:

ssize_t `H5Aget_name`(*hid_t* `attr_id`, *size_t* `buf_size`, *char* `*buf`)

Purpose:

Gets an attribute name.

Description:

`H5Aget_name` retrieves the name of an attribute specified by the identifier, `attr_id`. Up to `buf_size` characters are stored in buf followed by a \0 string terminator. If the name of the attribute is longer than (`buf_size` -1), the string terminator is stored in the last position of the buffer to properly terminate the string.

If the user only wants to find out the size of this name, the values `0` and `NULL` can be passed in for the parameters `bufsize` and `buf`.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute.
size_t `buf_size`	IN: The size of the buffer to store the name in.
char `*buf`	OUT: Buffer to store name in.

Returns:

Returns the length of the attribute's name, which may be longer than `buf_size`, if successful. Otherwise returns a negative value.

Fortran90 Interface: h5aget_name_f

```
SUBROUTINE h5aget_name_f(attr_id, size, buf, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id  ! Attribute identifier
  INTEGER(SIZE_T), INTENT(IN) :: size            ! Buffer size
  CHARACTER(LEN=*), INTENT(INOUT) :: buf
                                               ! Buffer to hold attribute name
  INTEGER, INTENT(OUT) :: hdferr         ! Error code:
                                         ! name length is successful,
                                         ! -1 if fail
END SUBROUTINE h5aget_name_f
```

Name: H5Aget_name_by_idx

Signature:

ssize_t `H5Aget_name_by_idx`(*hid_t* `loc_id`, *const char* *`obj_name`, *H5_index_t* `idx_type`, *H5_iter_order_t* `order`, *hsize_t* `n`, *char* *`name`, *size_t* `size`, *hid_t* `lapl_id`)

Purpose:

Gets an attribute name, by attribute index position

Description:

`H5Aget_name_by_idx` retrieves the name of an attribute that is attached to an object, which is specified by its location and name, `loc_id` and `obj_name`, respectively. The attribute is located by its index position, the size of the name is specified in `size`, and the attribute name is returned in `name`.

If `loc_id` fully specifies the object to which the attribute is attached, `obj_name` should be `'.'` (a dot).

The attribute is located by means of an index type, an index traversal order, and a position in the index, `idx_type`, `order` and `n`, respectively. These parameters and their valid values are discussed in the description of `H5Aiterate2`.

If the attribute name's size is unknown, the values `0` and `NULL` can be passed in for the parameters `size` and `name`. The function's return value will provide the correct value for `size`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See "Link Access Properties" in the H5P APIs.

Parameters:

hid_t `loc_id`	IN: Location of object to which attribute is attached
const char *`obj_name`	IN: Name of object to which attribute is attached, relative to location
H5_index_t `idx_type`	IN: Type of index
H5_iter_order_t `order`	IN: Index traversal order
hsize_t `n`	IN: Attribute's position in index
char *`name`	OUT: Attribute name
size_t `size`	IN: Size, in bytes, of attribute name
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns attribute name size, in bytes, if successful; otherwise returns a negative value.

Fortran90 Interface: h5aget_name_by_idx_f

```
SUBROUTINE h5aget_name_by_idx_f(loc_id, obj_name, idx_type, order, &
       n, name, hdferr, size, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id ! Identifer for object to which
                                       ! attribute is attached
  CHARACTER(LEN=*), INTENT(IN) :: obj_name ! Name of object, relative to
                                           ! location,from which attribute is to
                                           ! be removed
  INTEGER, INTENT(IN) :: idx_type
                               ! Type of index; Possible values are:
                               !  H5_INDEX_UNKNOWN_F   - Unknown index type
                               !  H5_INDEX_NAME_F      - Index on names
                               !  H5_INDEX_CRT_ORDER_F - Index on creation order
                               !  H5_INDEX_N_F         - Number indices defined
```

```
    INTEGER, INTENT(IN) :: order ! Order in which to iterate over index:
                                 !  H5_ITER_UNKNOWN_F - Unknown order
                                 !  H5_ITER_INC_F     - Increasing order
                                 !  H5_ITER_DEC_F     - Decreasing order
                                 !  H5_ITER_NATIVE_F  - No particular order,
                                 !                      whatever is fastest
                                 !  H5_ITER_N_F       - Number of iteration orders

    INTEGER(HSIZE_T), INTENT(IN) :: n
                                 ! Attribute’s position in index
    CHARACTER(LEN=*), INTENT(OUT) :: name
                                 ! Attribute name
    INTEGER, INTENT(OUT) :: hdferr
                                 ! Error code:
                                 !  Returns attribute name size,
                                 !  -1 if fail
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                 ! Link access property list
    INTEGER(SIZE_T), OPTIONAL, INTENT(OUT) :: size
                                 ! exact buffer size, in number of characters
  END SUBROUTINE h5aget_name_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aget_num_attrs

Signature:

int `H5Aget_num_attrs`(*hid_t* `loc_id`)

Purpose:

Determines the number of attributes attached to an object.

Description:

`H5Aget_num_attrs` returns the number of attributes attached to the object specified by its identifier, `loc_id`. The object can be a group, dataset, or named datatype.

Parameters:

hid_t `loc_id`	IN: Identifier of a group, dataset, or named datatype.

Returns:

Returns the number of attributes if successful; otherwise returns a negative value.

Fortran90 Interface: h5aget_num_attrs_f

```
SUBROUTINE h5aget_num_attrs_f(obj_id, attr_num, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id  ! Object identifier
  INTEGER, INTENT(OUT) :: attr_num      ! Number of attributes of the object
  INTEGER, INTENT(OUT) :: hdferr        ! Error code:
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5aget_num_attrs_f
```

Name: H5Aget_space

Signature:

hid_t `H5Aget_space`(*hid_t* `attr_id`)

Purpose:

Gets a copy of the dataspace for an attribute.

Description:

`H5Aget_space` retrieves a copy of the dataspace for an attribute. The dataspace identifier returned from this function must be released with `H5Sclose` or resource leaks will develop.

Parameters:

hid_t `attr_id`	IN: Identifier of an attribute.

Returns:

Returns attribute dataspace identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5aget_space_f

```
SUBROUTINE h5aget_space_f(attr_id, space_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id   ! Attribute identifier
  INTEGER(HID_T), INTENT(OUT) :: space_id ! Attribute dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code:
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5aget_space_f
```

Name: H5Aget_storage_size

Signature:

hsize_t `H5Aget_storage_size`(*hid_t* `attr_id`)

Purpose:

Returns the amount of storage required for an attribute.

Description:

`H5Aget_storage_size` returns the amount of storage that is required for the specified attribute, `attr_id`.

Parameters:

hid_t `attr_id`	IN: Identifier of the attribute to query.

Returns:

Returns the amount of storage size allocated for the attribute; otherwise returns 0 (zero).

Fortran90 Interface: h5aget_storage_size_f

```
SUBROUTINE h5aget_storage_size_f(attr_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id  ! Attribute identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: size  ! Attribute storage requirement
  INTEGER, INTENT(OUT) :: hdferr         ! Error code:
                                         !   0 on success and -1 on failure
END SUBROUTINE h5aget_storage_size_f
```

Name: H5Aget_type

Signature:

hid_t `H5Aget_type`(*hid_t* `attr_id`)

Purpose:

Gets an attribute datatype.

Description:

`H5Aget_type` retrieves a copy of the datatype for an attribute.

The datatype is reopened if it is a named type before returning it to the application. The datatypes returned by this function are always read-only. If an error occurs when atomizing the return datatype, then the datatype is closed.

The datatype identifier returned from this function must be released with `H5Tclose` or resource leaks will develop.

Parameters:

hid_t `attr_id`	IN: Identifier of an attribute.

Returns:

Returns a datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5aget_type_f

```
SUBROUTINE h5aget_type_f(attr_id, type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id  ! Attribute identifier
  INTEGER(HID_T), INTENT(OUT) :: type_id ! Attribute datatype identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code:
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5aget_type_f
```

Name: H5Aiterate

Signature:

herr_t `H5Aiterate`(*hid_t* `loc_id`, *unsigned* * `idx`, *H5A_operator_t* `op`, *void* *`op_data`) [1]

herr_t `H5Aiterate`(*hid_t* `obj_id`, *H5_index_t* `idx_type`, *H5_iter_order_t* `order`, *hsize_t* *`n`, *H5A_operator2_t* `op`, *void* *`op_data`) [2]

Purpose:

Calls a user's function for each attribute on an object.

Description:

`H5Aiterate` is a macro that is mapped to either `H5Aiterate1` or `H5Aiterate2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

- ◊ The `H5Aiterate` macro will be mapped to `H5Aiterate1` and will use the `H5Aiterate1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.
- ◊ The `H5Aiterate` macro mapped to `H5Aiterate2` and will use the `H5Aiterate2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in "API Compatibility Macros in HDF5"; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Aiterate` is mapped to the most recent version of the function, currently `H5Aiterate2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Aiterate` will be mapped to `H5Aiterate1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Aiterate` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Aiterate2`
Enable deprecated symbols	`H5Aiterate2`
Disable deprecated symbols	`H5Aiterate2`
Emulate Release 1.6 interface	`H5Aiterate1`
<u>Function-level macros</u>	
`H5Aiterate_vers = 2`	`H5Aiterate2`
`H5Aiterate_vers = 1`	`H5Aiterate1`

Interface history: Signature [1] above is the original `H5Aiterate` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Aiterate1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Aiterate2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	The function `H5Aiterate` renamed to `H5Aiterate1` and deprecated in this release. The macro `H5Aiterate` and the functions `H5Aiterate2` and `H5Aiterate_by_name` introduced in this release.

Name: H5Aiterate1
Signature:

herr_t `H5Aiterate1`(*hid_t* `loc_id`, *unsigned* * `idx`, *H5A_operator1_t* `op`, *void* *`op_data`)

Purpose:

Calls a user's function for each attribute on an object.

Notice:

This function is deprecated in favor of the function `H5Aiterate2`*.*

Description:

`H5Aiterate1` iterates over the attributes of the object specified by its identifier, `loc_id`. The object can be a group, dataset, or named datatype. For each attribute of the object, the `op_data` and some additional information specified below are passed to the operator function `op`. The iteration begins with the attribute specified by its index, `idx`; the index for the next attribute to be processed by the operator, `op`, is returned in `idx`. If `idx` is the null pointer, then all attributes are processed.

The prototype for `H5A_operator_t` is:

```
typedef herr_t (*H5A_operator1_t)(hid_t loc_id, const char *attr_name,
void *operator_data);
```

The operation receives the identifier for the group, dataset or named datatype being iterated over, `loc_id`, the name of the current attribute about the object, `attr_name`, and the pointer to the operator data passed in to `H5Aiterate1`, `op_data`. The return values from an operator are:

- ◊ Zero causes the iterator to continue, returning zero when all attributes have been processed.
- ◊ Positive causes the iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the next attribute.
- ◊ Negative causes the iterator to immediately return that value, indicating failure. The iterator can be restarted at the next attribute.

Parameters:

hid_t `loc_id`	IN: Identifier of a group, dataset or named datatype.
unsigned * `idx`	IN/OUT: Starting (IN) and ending (OUT) attribute index.
H5A_operator1_t `op`	IN: User's function to pass each attribute to
void *`op_data`	IN/OUT: User's data to pass through to iterator operator function

Returns:

If successful, returns the return value of the last operator if it was non-zero, or zero if all attributes were processed. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	The function `H5Aiterate` renamed to `H5Aiterate1` and deprecated in this release.

Name: H5Aiterate2

Signature:

herr_t `H5Aiterate2`(*hid_t* `obj_id`, *H5_index_t* `idx_type`, *H5_iter_order_t* `order`, *hsize_t* `*n`, *H5A_operator2_t* `op`, *void* `*op_data`,)

Purpose:

Calls user-defined function for each attribute on an object.

Description:

`H5Aiterate2` iterates over the attributes attached to a dataset, named datatype, or group, as specified by `obj_id`. For each attribute, user-provided data, `op_data`, with additional information as defined below, is passed to a user-defined function, `op`, which operates on that attribute.

The order of the iteration and the attributes iterated over are specified by three parameters: the index type, `idx_type`; the order in which the index is to be traversed, `order`; and the attribute's position in the index, `n`.

The type of index specified by `idx_type` can be one of the following:

`H5_INDEX_NAME`	An alpha-numeric index by attribute name
`H5_INDEX_CRT_ORDER`	An index by creation order

The order in which the index is to be traversed, as specified by `order`, can be one of the following:

`H5_ITER_INC`	Iteration is from beginning to end, i.e., a top-down iteration incrementing the index position at each step.
`H5_ITER_DEC`	Iteration starts at the end of the index, i.e., a bottom-up iteration decrementing the index position at each step.
`H5_ITER_NATIVE`	HDF5 iterates in the fastest-available order. No information is provided as to the order, but HDF5 ensures that each element in the index will be visited if the iteration completes successfully.

The next attribute to be operated on is specified by `n`, a position in the index.

For example, if `idx_type`, `order`, and `n` are set to `H5_INDEX_NAME`, `H5_ITER_INC`, and `5`, respectively, the attribute in question is the fifth attribute from the beginning of the alpha-numeric index of attribute names. If `order` were set to `H5_ITER_DEC`, it would be the fifth attribute from the end of the index.

The parameter `n` is passed in on an `H5Aiterate2` call with one value and may be returned with another value. The value passed in identifies the parameter to be operated on first; the value returned identifies the parameter to be operated on in the next step of the iteration.

The `H5A_operator2_t` prototype for the `op` parameter is as follows:

typedef herr_t (`*H5A_operator2_t`)(*hid_t* `location_id/*in*/`, *const char* `*attr_name/*in*/`, *const H5A_info_t* `*ainfo/*in*/`, *void* `*op_data/*in,out*/`)

The operation receives the location identifier for the group or dataset being iterated over, `location_id`; the name of the current object attribute, `attr_name`; the attribute's *info* struct, `ainfo`; and a pointer to the operator data passed into `H5Aiterate2`, `op_data`.

Valid return values from an operator and the resulting `H5Aiterate2` and `op` behavior are as follows:

- ◊ Zero causes the iterator to continue, returning zero when all attributes have been processed.
- ◊ A positive value causes the iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the next attribute, as indicated by the return value of `n`.
- ◊ A negative value causes the iterator to immediately return that value, indicating failure. The iterator can be restarted at the next attribute, as indicated by the return value of `n`.

Parameters:

hid_t `obj_id`	IN: Identifier for object to which attributes are attached; may be group, dataset, or named datatype.
H5_index_t `idx_type`	IN: Type of index
H5_iter_order_t `order`	IN: Order in which to iterate over index
hsize_t `*n`	IN/OUT: Initial and returned offset within index
H5A_operator2_t `op`	IN: User-defined function to pass each attribute to
void `*op_data`	IN/OUT: User data to pass through to and to be returned by iterator operator function

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Further note that this function returns the return value of the last operator if it was non-zero, which can be a negative value, zero if all attributes were processed, or a positive value indicating short-circuit success (see above).

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aiterate_by_name

Signature:

herr_t `H5Aiterate_by_name`(*hid_t* `loc_id`, *const char* *`obj_name`, *H5_index_t* `idx_type`, *H5_iter_order_t* `order`, *hsize_t* *`n`, *H5A_operator2_t* `op`, *void* *`op_data`, *hid_t* `lapd_id`)

Purpose:

Calls user-defined function for each attribute on an object.

Description:

`H5Aiterate_by_name` iterates over the attributes attached to the dataset or group specified with `loc_id` and `obj_name`. For each attribute, user-provided data, `op_data`, with additional information as defined below, is passed to a user-defined function, `op`, which operates on that attribute.

If `loc_id` fully specifies the object to which these attributes are attached, `obj_name` should be `'.'` (a dot).

The order of the iteration and the attributes iterated over are specified by three parameters: the index type, `idx_type`; the order in which the index is to be traversed, `order`; and the attribute's position in the index, `n`.

The type of index specified by `idx_type` can be one of the following:

`H5_INDEX_NAME`	An alpha-numeric index by attribute name
`H5_INDEX_CRT_ORDER`	An index by creation order

The order in which the index is to be traversed, as specified by `order`, can be one of the following:

`H5_ITER_INC`	Iteration is from beginning to end, i.e., a top-down iteration incrementing the index position at each step.
`H5_ITER_DEC`	Iteration starts at the end of the index, i.e., a bottom-up iteration decrementing the index position at each step.
`H5_ITER_NATIVE`	HDF5 iterates in the fastest-available order. No information is provided as to the order, but HDF5 ensures that each element in the index will be visited if the iteration completes successfully.

The next attribute to be operated on is specified by `n`, a position in the index.

For example, if `idx_type`, `order`, and `n` are set to `H5_INDEX_NAME`, `H5_ITER_INC`, and `5`, respectively, the attribute in question is the fifth attribute from the beginning of the alpha-numeric index of attribute names. If `order` were set to `H5_ITER_DEC`, it would be the fifth attribute from the end of the index.

The parameter `n` is passed in on an `H5Aiterate_by_name` call with one value and may be returned with another value. The value passed in identifies the parameter to be operated on first; the value returned identifies the parameter to be operated on in the next step of the iteration.

The `H5A_operator2_t` prototype for the `op` parameter is as follows:

typedef herr_t (*`H5A_operator2_t`)(*hid_t* `location_id/*in*/`, *const char* *`attr_name/*in*/`, *const H5A_info_t* *`ainfo/*in*/`, *void* *`op_data/*in,out*/`)

The operation receives the location identifier for the group or dataset being iterated over, `location_id`; the name of the current object attribute, `attr_name`; the attribute's *info* struct, `ainfo`; and a pointer to the operator data passed into `H5Aiterate_by_name`, `op_data`.

Valid return values from an operator and the resulting `H5Aiterate_by_name` and `op` behavior are as follows:

- ◊ Zero causes the iterator to continue, returning zero when all attributes have been processed.
- ◊ A positive value causes the iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the next attribute, as indicated by the return value of `n`.
- ◊ A negative value causes the iterator to immediately return that value, indicating failure. The iterator can be restarted at the next attribute, as indicated by the return value of `n`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See "Link Access Properties" in the H5P APIs.

Parameters:

hid_t `loc_id`	IN: Location or object identifier; may be dataset or group
const char *`obj_name`	IN: Name of object, relative to location
H5_index_t `idx_type`	IN: Type of index
H5_iter_order_t `order`	IN: Order in which to iterate over index
hsize_t *`n`	IN/OUT: Initial and returned offset within index
H5A_operator2_t `op`	IN: User-defined function to pass each attribute to
void *`op_data`	IN/OUT: User data to pass through to and to be returned by iterator operator function
hid_t `lapd_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Further note that this function returns the return value of the last operator if it was non-zero, which can be a negative value, zero if all attributes were processed, or a positive value indicating short-circuit success (see above).

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aopen

Signature:

herr_t `H5Aopen`(*hid_t* `obj_id`, *const char* *`attr_name`, *hid_t* `aapl_id`)

Purpose:

Opens an attribute for an object specified by object identifier and attribute name.

Description:

`H5Aopen` opens an existing attribute, `attr_name`, that is attached to an object specified an object identifier, `object_id`.

The attribute access property list, `aapl_id`, is currently unused and should currently be `H5P_DEFAULT`.

This function, `H5Aopen_by_idx`, or `H5Aopen_by_name` must be called before an attribute can be accessed for any further purpose, including reading, writing, or any modification.

The attribute identifier returned by this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `obj_id`	IN: Identifer for object to which attribute is attached
const char *`attr_name`	IN: Name of attribute to open
hid_t `aapl_id`	IN: Attribute access property list

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5aopen_f

```
SUBROUTINE h5aopen_f(obj_id, attr_name, attr_id, hdferr, aapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id      ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: attr_name ! Attribute name
  INTEGER(HID_T), INTENT(OUT) :: attr_id    ! Attribute identifier
  INTEGER, INTENT(OUT) :: hdferr            ! Error code:
                                            ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: aapl_id
                                            ! Attribute access property list
END SUBROUTINE h5aopen_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aopen_by_idx

Signature:

herr_t `H5Aopen_by_idx`(*hid_t* `loc_id`, *const char* *`obj_name`, *H5_index_t* `idx_type`, *H5_iter_order_t* `order`, *hsize_t* `n`, *hid_t* `aapl_id`, *hid_t* `lapl_id`)

Purpose:

Description:

`H5Aopen_by_idx` opens an existing attribute that is attached to an object specified by location and name, `loc_id` and `obj_name`, respectively. If `loc_id` fully specifies the object to which the attribute is attached, `obj_name` should be `'.'` (a dot).

The attribute is identified by an index type, an index traversal order, and a position in the index, `idx_type`, `order` and `n`, respectively. These parameters and their valid values are discussed in the description of `H5Aiterate2`.

The attribute access property list, `aapl_id`, is currently unused and should currently be `H5P_DEFAULT`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

This function, `H5Aopen`, or `H5Aopen_by_name` must be called before an attribute can be accessed for any further purpose, including reading, writing, or any modification.

The attribute identifier returned by this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Location of object to which attribute is attached
const char *`obj_name`	IN: Name of object to which attribute is attached, relative to location
H5_index_t `idx_type`	IN: Type of index
H5_iter_order_t `order`	IN: Index traversal order
hsize_t `n`	IN: Attribute’s position in index
hid_t `aapl_id`	IN: Attribute access property list
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5aopen_by_idx_f

```
SUBROUTINE h5aopen_by_idx_f(loc_id, obj_name, idx_type, order, n, attr_id, &
     hdferr, aapl_id, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                              ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                              ! Name of object to which attribute is attached
  INTEGER, INTENT(IN) :: idx_type
                              ! Type of index; Possible values are:
                              !  H5_INDEX_UNKNOWN_F   - Unknown index type
                              !  H5_INDEX_NAME_F      - Index on names
                              !  H5_INDEX_CRT_ORDER_F - Index on creation order
                              !  H5_INDEX_N_F         - Number of indices defined
```

```
    INTEGER, INTENT(IN) :: order
                                ! Order in which to iterate over index:
                                !  H5_ITER_UNKNOWN_F  - Unknown order
                                !  H5_ITER_INC_F      - Increasing order
                                !  H5_ITER_DEC_F      - Decreasing order
                                !  H5_ITER_NATIVE_F   - No particular order,
                                !                       whatever is fastest

    INTEGER(HSIZE_T), INTENT(IN) :: n
                                ! Attribute’s position in index
    INTEGER(HID_T), INTENT(OUT) :: attr_id
                                ! Attribute identifier
    INTEGER, INTENT(OUT) :: hdferr
                                ! Error code:
                                ! 0 on success and -1 on failure
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: aapl_id
                                ! Attribute access property list
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                ! Link access property list
  END SUBROUTINE h5aopen_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aopen_by_name

Signature:

herr_t `H5Aopen_by_name`(*hid_t* `loc_id`, *const char* *`obj_name`, *const char* *`attr_name`, *hid_t* `aapl_id`, *hid_t* `lapl_id`)

Purpose:

Opens an attribute for an object by object name and attribute name.

Description:

`H5Aopen_by_name` opens an existing attribute, `attr_name`, that is attached to an object specified by location and name, `loc_id` and `obj_name`, respectively.

`loc_id` specifies a location from which the target object can be located and `obj_name` is an object name relative to `loc_id`. If `loc_id` fully specifies the object to which the attribute is attached, `obj_name` should be `'.'` (a dot).

The attribute access property list, `aapl_id`, is currently unused and should currently be `H5P_DEFAULT`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

This function, `H5Aopen`, or `H5Aopen_by_idx` must be called before an attribute can be accessed for any further purpose, including reading, writing, or any modification.

The attribute identifier returned by this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Location from which to find object to which attribute is attached
const char *`obj_name`	IN: Name of object to which attribute is attached, relative to `loc_id`
const char *`attr_name`	IN: Name of attribute to open
hid_t `aapl_id`	IN: Attribute access property list *(Currently unused; should be passed in as `H5P_DEFAULT`.)*
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns an attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE h5aopen_by_name_f(loc_id, obj_name, attr_name, attr_id, hdferr, &
      aapl_id, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                                     ! Location identifier
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                                     ! Object name either relative to loc_id,
                                     ! absolute from file’s root group, or '.'
  CHARACTER(LEN=*), INTENT(IN) :: attr_name
                                     ! Attribute name
  INTEGER(HID_T), INTENT(OUT) :: attr_id
                                     ! Attribute identifier
  INTEGER, INTENT(OUT) :: hdferr     ! Error code:
                                     ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: aapl_id
                                     ! Attribute access property list
```

```
                                            ! (Currently unused; set to H5P_DEFAULT_F)
          INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                            ! Link access property list identifier
        END SUBROUTINE
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Aopen_idx

Signature:

hid_t `H5Aopen_idx`(*hid_t* `loc_id`, *unsigned int* `idx`)

Purpose:

Opens the attribute specified by its index.

Description:

`H5Aopen_idx` opens an attribute which is attached to the object specified with `loc_id`. The location object may be either a group, dataset, or named datatype, all of which may have any sort of attribute. The attribute specified by the index, `idx`, indicates the attribute to access. The value of `idx` is a 0-based, non-negative integer. The attribute identifier returned from this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Identifier of the group, dataset, or named datatype attribute to be attached to.
unsigned int `idx`	IN: Index of the attribute to open.

Returns:

Returns attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5aopen_idx_f

```
SUBROUTINE h5aopen_idx_f(obj_id, index, attr_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id    ! Object identifier
  INTEGER, INTENT(IN) :: index            ! Attribute index
  INTEGER(HID_T), INTENT(OUT) :: attr_id  ! Attribute identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code:
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5aopen_idx_f
```

Name: H5Aopen_name

Signature:

hid_t `H5Aopen_name`(*hid_t* `loc_id`, *const char* *`name`)

Purpose:

Opens an attribute specified by name.

Description:

`H5Aopen_name` opens an attribute specified by its name, `name`, which is attached to the object specified with `loc_id`. The location object may be either a group, dataset, or named datatype, which may have any sort of attribute. The attribute identifier returned from this function must be released with `H5Aclose` or resource leaks will develop.

Parameters:

hid_t `loc_id`	IN: Identifier of a group, dataset, or named datatype that attribute is attached to.
const char *`name`	IN: Attribute name.

Returns:

Returns attribute identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5aopen_name_f

```
SUBROUTINE h5aopen_name_f(obj_id, name, attr_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id    ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: name    ! Attribute name
  INTEGER(HID_T), INTENT(OUT) :: attr_id  ! Attribute identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code:
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5aopen_name_f
```

Name: H5Aread
Signature:
herr_t `H5Aread`(*hid_t* `attr_id`, *hid_t* `mem_type_id`, *void* *`buf`)
Purpose:
Reads an attribute.
Description:
`H5Aread` reads an attribute, specified with `attr_id`. The attribute's memory datatype is specified with `mem_type_id`. The entire attribute is read into `buf` from the file.

Datatype conversion takes place at the time of a read or write and is automatic. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.
Parameters:

hid_t `attr_id`	IN: Identifier of an attribute to read.
hid_t `mem_type_id`	IN: Identifier of the attribute datatype (in memory).
void *`buf`	OUT: Buffer for data to be read.

Returns:
Returns a non-negative value if successful; otherwise returns a negative value.
Fortran90 Interface: h5aread_f
```
SUBROUTINE h5aread_f(attr_id, memtype_id,  buf, dims, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id    ! Attribute identifier
  INTEGER(HID_T), INTENT(IN) :: memtype_id ! Attribute datatype
                                           ! identifier  (in memory)
  TYPE, INTENT(INOUT)  :: buf              ! Data buffer; may be a scalar or
                                           ! an array
  DIMENSION(*), INTEGER(HSIZE_T), INTENT(IN)  :: dims
                                           ! Array to hold corresponding
                                           ! dimension sizes of data buffer buf;
                                           ! dim(k) has value of the
                                           ! k-th dimension of buffer buf;
                                           ! values are ignored if buf is a
                                           ! scalar
  INTEGER, INTENT(OUT) :: hdferr           ! Error code:
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5aread_f
```

History:

Release	**Fortran90**
1.4.2	The `dims` parameter was added in this release.

Name: H5Arename

Signature:

herr_t `H5Arename`(*hid_t* `loc_id`, *char* *`old_attr_name`, *char* *`new_attr_name`)

Purpose:

Renames an attribute.

Description:

`H5Arename` changes the name of the attribute located at `loc_id`.

The old name, `old_attr_name`, is changed to the new name, `new_attr_name`.

Parameters:

hid_t `loc_id`	IN: Location of the attribute.
char *`old_attr_name`	IN: Name of the attribute to be changed.
char *`new_attr_name`	IN: New name for the attribute.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5arename_f

```
SUBROUTINE h5arename_f(loc_id, old_attr_name, new_attr_name, hdferr)
    IMPLICIT NONE
    INTEGER(HID_T), INTENT(IN) :: loc_id          ! Object identifier
    CHARACTER(LEN=*), INTENT(IN) :: old_attr_name ! Prior attribute name
    CHARACTER(LEN=*), INTENT(IN) :: new_attr_name ! New attribute name
    INTEGER, INTENT(OUT) :: hdferr                ! Error code:
                                                  !  0 on success, -1 on failure
END SUBROUTINE h5arename_f
```

Name: H5Arename_by_name

Signature:

herr_t `H5Arename_by_name`(*hid_t* `loc_id`, *const char* *`obj_name`, *const char* *`old_attr_name`, *const char* *`new_attr_name`, *hid_t* `lapl_id`)

Purpose:

Renames an attribute.

Description:

`H5Arename_by_name` changes the name of attribute that is attached to the object specified by `loc_id` and `obj_name`. The attribute named `old_attr_name` is renamed `new_attr_name`.

The link access property list, `lapl_id`, may provide information regarding the properties of links required to access the object, `obj_name`. See “Link Access Properties” in the H5P APIs.

Parameters:

hid_t `loc_id`	IN: Location or object identifier; may be dataset or group
const char *`obj_name`	IN: Name of object, relative to location, whose attribute is to be renamed
const char *`old_attr_name`	IN: Prior attribute name
const char *`new_attr_name`	IN: New attribute name
hid_t `lapl_id`	IN: Link access property list identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5arename_by_name_f

```
SUBROUTINE h5arename_by_name_f(loc_id, obj_name, old_attr_name, new_attr_name, &
     hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id  ! Object identifier
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                                        ! Name of object, relative to location,
                                        ! whose attribute is to be renamed
  CHARACTER(LEN=*), INTENT(IN) :: old_attr_name
                                        ! Prior attribute name
  CHARACTER(LEN=*), INTENT(IN) :: new_attr_name
                                        ! New attribute name
  INTEGER, INTENT(OUT) :: hdferr        ! Error code:
                                        ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                        ! Link access property list identifier
END SUBROUTINE h5arename_by_name_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Awrite
Signature:
herr_t `H5Awrite`(*hid_t* `attr_id`, *hid_t* `mem_type_id`, *const void* *`buf`)
Purpose:
Writes data to an attribute.
Description:
`H5Awrite` writes an attribute, specified with `attr_id`. The attribute's memory datatype is specified with `mem_type_id`. The entire attribute is written from `buf` to the file.

Datatype conversion takes place at the time of a read or write and is automatic. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.
Parameters:

hid_t `attr_id`	IN: Identifier of an attribute to write.
hid_t `mem_type_id`	IN: Identifier of the attribute datatype (in memory).
const void *`buf`	IN: Data to be written.

Returns:
Returns a non-negative value if successful; otherwise returns a negative value.
Fortran90 Interface: h5awrite_f

```
SUBROUTINE h5awrite_f(attr_id, memtype_id,  buf, dims, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: attr_id    ! Attribute identifier
  INTEGER(HID_T), INTENT(IN) :: memtype_id ! Attribute datatype
                                           ! identifier  (in memory)
  TYPE, INTENT(IN) :: buf                  ! Data buffer; may be a scalar or
                                           ! an array
  DIMENSION(*), INTEGER(HSIZE_T), INTENT(IN)  :: dims
                                           ! Array to hold corresponding
                                           ! dimension sizes of data buffer buf;
                                           ! dim(k) has value of the k-th
                                           ! dimension of buffer buf;
                                           ! values are ignored if buf is
                                           ! a scalar
  INTEGER, INTENT(OUT) :: hdferr           ! Error code:
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5awrite_f
```

History:

Release	**Fortran90**
1.4.2	The `dims` parameter was added in this release.

H5D: Datasets Interface

Dataset Object API Functions

These functions create and manipulate dataset objects, and set and retrieve their constant or persistent properties.

The C Interfaces:

- H5Dcreate
- H5Dcreate1 *
- H5Dcreate2
- H5Dcreate_anon
- H5Dopen
- H5Dopen1 *
- H5Dopen2
- H5Dclose

- H5Dget_space
- H5Dget_space_status
- H5Dget_type
- H5Dget_create_plist
- H5Dget_access_plist
- H5Dget_offset
- H5Dget_storage_size

- H5Dvlen_get_buf_size
- H5Dvlen_reclaim
- H5Dread
- H5Dwrite
- H5Diterate
- H5Dextend *
- H5Dset_extent
- H5Dfill

** Use of these functions is deprecated in Release 1.8.0.*

Alphabetical Listing

- H5Dclose
- H5Dcreate
- H5Dcreate1 *
- H5Dcreate2
- H5Dcreate_anon
- H5Dextend *
- H5Dfill

- H5Dget_access_plist
- H5Dget_create_plist
- H5Dget_offset
- H5Dget_space
- H5Dget_space_status
- H5Dget_storage_size
- H5Dget_type
- H5Diterate

- H5Dopen
- H5Dopen1 *
- H5Dopen2
- H5Dread
- H5Dset_extent
- H5Dvlen_get_buf_size
- H5Dvlen_reclaim
- H5Dwrite

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5dclose_f
- h5dcreate_f
- h5dcreate_anon_f
- h5dopen_f
- h5dget_space_f
- h5dget_space_status_f

- h5dget_type_f
- h5dget_create_plist_f
- h5dget_offset_f
- h5dget_storage_size_f
- h5dvlen_get_max_len_f

- h5dread_f
- h5dread_vl_f
- h5dwrite_f
- h5dwrite_vl_f
- h5dextend_f
- h5dfill_f

Name: H5Dclose

Signature:

herr_t `H5Dclose`(*hid_t* `dataset_id`)

Purpose:

Closes the specified dataset.

Description:

`H5Dclose` ends access to a dataset specified by `dataset_id` and releases resources used by it. Further use of the dataset identifier is illegal in calls to the dataset API.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to close access to.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5dclose_f

```
SUBROUTINE h5dclose_f(dset_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id ! Dataset identifier
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5dclose_f
```

Name: H5Dcreate

Signature:

hid_t `H5Dcreate`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `dcpl_id`)

hid_t `H5Dcreate`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `dtype_id`, *hid_t* `space_id`, *hid_t* `lcpl_id`, *hid_t* `dcpl_id`, *hid_t* `dapl_id`)

Purpose:

Creates a new dataset and links it to a location in the file.

Description:

`H5Dcreate` is a macro that is mapped to either `H5Dcreate1` or `H5Dcreate2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Dcreate` is mapped to the most recent version of the function, currently `H5Dcreate2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Dcreate` will be mapped to `H5Dcreate1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Dcreate` mapping**
Global settings	
No compatibility flag	`H5Dcreate2`
Enable deprecated symbols	`H5Dcreate2`
Disable deprecated symbols	`H5Dcreate2`
Emulate Release 1.6 interface	`H5Dcreate1`
Function-level macros	
`H5Dcreate_vers = 2`	`H5Dcreate2`
`H5Dcreate_vers = 1`	`H5Dcreate1`

Fortran90 Interface: h5dcreate_f

```
SUBROUTINE h5dcreate_f(loc_id, name, type_id, space_id, dset_id, &
     hdferr, dcpl_id, lcpl_id, dapl_id)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the dataset
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  INTEGER(HID_T), INTENT(IN) :: space_id ! Dataspace identifier
  INTEGER(HID_T), INTENT(OUT) :: dset_id ! Dataset identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
```

```
                                          ! 0 on success and -1 on failure
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: dcpl_id
                                          ! Dataset creation property list
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                                          ! Link creation property list
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: dapl_id
                                          ! Dataset access property list
  END SUBROUTINE h5dcreate_f
```

History:

Release	**C**
1.8.0	The function `H5Dcreate` renamed to `H5Dcreate1` and deprecated in this release. The macro `H5Dcreate` and the function `H5Dcreate2` introduced in this release.

Last modified: 18 May 2009

Name: H5Dcreate1

Signature:

hid_t `H5Dcreate1`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `dcpl_id`)

Purpose:

Creates a dataset at the specified location.

Notice:

This function is deprecated in favor of the function `H5Dcreate2` or the macro `H5Dcreate`.

Description:

`H5Dcreate1` creates a data set with a name, `name`, in the file or in the group specified by the identifier `loc_id`.

`name` can be a relative path based at `loc_id` or an absolute path from the root of the file. If any of the groups specified in that path do not already exist, the dataset must be created with `H5Dcreate_anon` and linked into the file structure with `H5Lcreate_hard`.

The dataset's datatype and dataspace are specified by `type_id` and `space_id`, respectively. These are the datatype and dataspace of the dataset as it will exist in the file, which may differ from the datatype and dataspace in application memory.

Names within a group are unique: `H5Dcreate1` will return an error if a link with the name specified in `name` already exists at the location specified in `loc_id`.

As is the case for any object in a group, the length of a dataset name is not limited.

`create_plist_id` is an `H5P_DATASET_CREATE` property list created with `H5Pcreate1` and initialized with various property list functions described in "H5P: Property List Interface."

`H5Dcreate` and `H5Dcreate_anon` return an error if the dataset's datatype includes a variable-length (VL) datatype and the fill value is undefined, i.e., set to `NULL` in the dataset creation property list. Such a VL datatype may be directly included, indirectly included as part of a compound or array datatype, or indirectly included as part of a nested compound or array datatype.

`H5Dcreate` and `H5Dcreate_anon` return a dataset identifier for success or a negative value for failure. The dataset identifier should eventually be closed by calling `H5Dclose` to release resources it uses.

See `H5Dcreate_anon` for discussion of the differences between `H5Dcreate` and `H5Dcreate_anon`.

Fill values and space allocation:

The HDF5 library provides flexible means of specifying a fill value, of specifying when space will be allocated for a dataset, and of specifying when fill values will be written to a dataset. For further information on these topics, see the document *Fill Value and Dataset Storage Allocation Issues in HDF5* and the descriptions of the following HDF5 functions in this *HDF5 Reference Manual*:

```
H5Dfill                 H5Pset_fill_time
H5Pset_fill_value       H5Pget_fill_time
```

`H5Pget_fill_value`	`H5Pset_alloc_time`
`H5Pfill_value_defined`	`H5Pget_alloc_time`

This information is also included in the “HDF5 Datasets” chapter of the new *HDF5 User's Guide*, which is being prepared for release.

Note:

`H5Dcreate` and `H5Dcreate_anon` can fail if there has been an error in setting up an element of the dataset creation property list. In such cases, each item in the property list must be examined to ensure that the setup satisfies all required conditions. This problem is most likely to occur with the use of filters.

For example, either function will fail without a meaningful explanation if the following conditions exist simultaneously:

◊ SZIP compression is being used on the dataset.
◊ The SZIP parameter `pixels_per_block` is set to an inappropriate value.

In such a case, one would refer to the description of `H5Pset_szip`, looking for any conditions or requirements that might affect the local computing environment.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group within which to create the dataset.
const char * `name`	IN: The name of the dataset to create.
hid_t `type_id`	IN: Identifier of the datatype to use when creating the dataset.
hid_t `space_id`	IN: Identifier of the dataspace to use when creating the dataset.
hid_t `create_plist_id`	IN: Identifier of the set creation property list.

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Dcreate`.

History:

Release	**C**
1.8.0	Function `H5Dcreate` renamed to `H5Dcreate1` and deprecated in this release.

Name: H5Dcreate2

Signature:

hid_t `H5Dcreate2`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `dtype_id`, *hid_t* `space_id`, *hid_t* `lcpl_id`, *hid_t* `dcpl_id`, *hid_t* `dapl_id`)

Purpose:

Creates a new dataset and links it into the file.

Description:

`H5Dcreate2` creates a new dataset named `name` at the location specified by `loc_id`, and associates constant and initial persistent properties with that dataset, including `dtype_id`, the datatype of each data element as stored in the file; `space_id`, the dataspace of the dataset; and other initial properties as defined in the dataset creation property and access property lists, `dcpl_id` and `dapl_id`, respectively. Once created, the dataset is opened for access.

`loc_id` may be a file identifier, or a group identifier within that file. `name` may be either an absolute path in the file or a relative path from `loc_id` naming the dataset.

The link creation property list, `lcpl_id`, governs creation of the link(s) by which the new dataset is accessed and the creation of any intermediate groups that may be missing.

The datatype and dataspace properties and the dataset creation and access property lists are attached to the dataset, so the caller may derive new datatypes, dataspaces, and creation and access properties from the old ones and reuse them in calls to create additional datasets.

Once created, the dataset is ready to receive raw data. Immediately attempting to read raw data from the dataset will probably return the fill value.

To conserve and release resources, the dataset should be closed when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier
const char *`name`	IN: Dataset name
hid_t `dtype_id`	IN: Datatype identifier
hid_t `space_id`	IN: Dataspace identifier
hid_t `lcpl_id`	IN: Link creation property list
hid_t `dcpl_id`	IN: Dataset creation property list
hid_t `dapl_id`	IN: Dataset access property list

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Dcreate`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Last modified: 18 May 2009

Name: H5Dcreate_anon

Signature:

hid_t `H5Dcreate_anon`(*hid_t* `loc_id`, *hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `dcpl_id`, *hid_t* `dapl_id`)

Purpose:

Creates a dataset in a file without linking it into the file structure.

Description:

`H5Dcreate_anon` creates a dataset in the file specified by `loc_id`.

`loc_id` may be a file identifier or a group identifier within that file.

The dataset's datatype and dataspace are specified by `type_id` and `space_id`, respectively. These are the datatype and dataspace of the dataset as it will exist in the file, which may differ from the datatype and dataspace in application memory.

Dataset creation properties are specified in the dataset creation property list `dcpl_id`. Dataset access properties are specified in the dataset access property list `dapl_id`.

`H5Dcreate_anon` returns a new dataset identifier. Using this identifier, the new dataset *must* be linked into the HDF5 file structure with `H5Lcreate_hard` or it will be deleted from the file when the file is closed.

See `H5Dcreate` for further details and considerations on the use of `H5Dcreate` and `H5Dcreate_anon`.

The differences between this function and `H5Dcreate` are as follows:

- ◊ `H5Dcreate_anon` explicitly includes a dataset access property list. `H5Dcreate` always uses default dataset access properties.
- ◊ `H5Dcreate_anon` neither provides the new dataset's name nor links it into the HDF5 file structure; those actions must be performed separately through a call to `H5Lcreate_hard`, which offers greater control over linking.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group within which to create the dataset.
hid_t `type_id`	IN: Identifier of the datatype to use when creating the dataset.
hid_t `space_id`	IN: Identifier of the dataspace to use when creating the dataset.
hid_t `dcpl_id`	IN: Dataset creation property list identifier.
hid_t `dapl_id`	IN: Dataset access property list identifier.

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE h5dcreate_anon_f(loc_id, type_id, space_id, dset_id, hdferr, &
                            dcpl_id, dapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: loc_id   ! File or group identifier.
  INTEGER(HID_T), INTENT(IN)  :: type_id  ! Datatype identifier.
  INTEGER(HID_T), INTENT(IN)  :: space_id ! Dataspace identifier.
  INTEGER(HID_T), INTENT(OUT) :: dset_id  ! Dataset identifier.
```

```
        INTEGER, INTENT(OUT) :: hdferr          ! Error code.
                                                ! 0 on success and -1 on failure
        INTEGER(HID_T), OPTIONAL, INTENT(IN) :: dcpl_id
                                                ! Dataset creation property list
                                                ! identifier.
        INTEGER(HID_T), OPTIONAL, INTENT(IN) :: dapl_id
                                                ! Dataset access property list
                                                ! identifier.
      END SUBROUTINE h5dcreate_anon_f
```

Name: H5Dextend
Signature:
herr_t `H5Dextend`(*hid_t* `dataset_id`, *const hsize_t* * `size`)
Purpose:
Extends a dataset.
Notice:
This function is deprecated in favor of the function `H5Dset_extent`*.*
Description:
`H5Dextend` verifies that the dataset is at least of size `size`, extending it if necessary. The dimensionality of `size` is the same as that of the dataspace of the dataset being changed.

This function can be applied to the following datasets:

◊ Any dataset with unlimited dimensions
◊ A dataset with fixed dimensions if the current dimension sizes are less than the maximum sizes set with `maxdims` (see `H5Screate_simple`)

Space on disk is immediately allocated for the new dataset extent if the dataset’s space allocation time is set to `H5D_ALLOC_TIME_EARLY`. Fill values will be written to the dataset if the dataset’s fill time is set to `H5D_FILL_TIME_IFSET` or `H5D_FILL_TIME_ALLOC`. (See `H5Pset_fill_time` and `H5Pset_alloc_time`.)

This function ensures that the dataset dimensions are of at least the sizes specified in `size`. The function `H5Dset_extent` must be used if the dataset dimension sizes are are to be reduced.
Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset.
const hsize_t * `size`	IN: Array containing the new magnitude of each dimension.

Returns:
Returns a non-negative value if successful; otherwise returns a negative value.
Fortran90 Interface: h5dextend_f
```
SUBROUTINE h5dextend_f(dataset_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dataset_id   ! Dataset identifier
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN)  :: size
                                             ! Array containing
                                             ! dimensions' sizes
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5dextend_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Dfill

Signature:

herr_t `H5Dfill`(*const void* *`fill`, *hid_t* `fill_type_id`, *void* *`buf`, *hid_t* `buf_type_id`, *hid_t* `space_id`)

Purpose:

Fills dataspace elements with a fill value in a memory buffer.

Description:

`H5Dfill` explicitly fills the dataspace selection in memory, `space_id`, with the fill value specified in `fill`. If `fill` is `NULL`, a fill value of `0` (zero) is used.

`fill_type_id` specifies the datatype of the fill value.
`buf` specifies the buffer in which the dataspace elements will be written.
`buf_type_id` specifies the datatype of those data elements.

Note that if the fill value datatype differs from the memory buffer datatype, the fill value will be converted to the memory buffer datatype before filling the selection.

Note:

Applications sometimes write data only to portions of an allocated dataset. It is often useful in such cases to fill the unused space with a known fill value. See H5Pset_fill_value for further discussion. Other related functions include H5Pget_fill_value, H5Pfill_value_defined, H5Pset_fill_time, H5Pget_fill_time, H5Dcreate, and H5Dcreate_anon.

Parameters:

const void *`fill`	IN: Pointer to the fill value to be used.
hid_t `fill_type_id`	IN: Fill value datatype identifier.
void *`buf`	IN/OUT: Pointer to the memory buffer containing the selection to be filled.
hid_t `buf_type_id`	IN: Datatype of dataspace elements to be filled.
hid_t `space_id`	IN: Dataspace describing memory buffer and containing the selection to be filled.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5dfill_f

```
SUBROUTINE h5dfill_f(fill_value, space_id, buf, hdferr)
  IMPLICIT NONE
  TYPE, INTENET(IN) :: fill_value        ! Fill value; may be have one of the
                                         ! following types:
                                         ! INTEGER, REAL, DOUBLE PRECISION,
                                         ! CHARACTER
  INTEGER(HID_T), INTENT(IN) :: space_id ! Memory dataspace selection identifier
  TYPE, DIMENSION(*) :: buf              ! Memory buffer to fill in; must have
                                         ! the same datatype as fill value
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5dfill_f
```

Last modified: 8 May 2009

Name: H5Dget_access_plist

Signature:

hid_t `H5Dget_access_plist(` *hid_t* `dataset_id`)

Purpose:

Returns the dataset access property list associated with a dataset.

Description:

`H5Dget_access_plist` returns a copy of the dataset access property list used to open the specified dataset. Modifications to the returned property list will have no effect on the dataset it was retrieved from.

The chunk cache parameters in the returned property lists will be those used by the dataset. If the properties in the file access property list were used to determine the dataset's chunk cache configuration, then those properties will be present in the returned dataset access property list. If the dataset does not use a chunked layout, then the chunk cache properties will be set to the default. The chunk cache properties in the returned list are considered to be “set”, and any use of this list will override the corresponding properties in the fileâ’ s file access property list.

All link access properties in the returned list will be set to the default values.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to get access property list of.

Returns:

Returns a dataset access property list identifier if successful; otherwise returns a negative value.

Example Usage:

The following code retrieves the dataset access property list used to open the dataset `dataset_id` into `dapl_id`:

```
dapl_id = H5Dget_access_plist(dataset_id);
```

See Also:

“Dataset Access Properties” in the “H5P: Property List Interface” chapter of the *HDF5 Reference Manual*

History:

Release	**Change**
1.8.3	C function introduced in this release.

Name: H5Dget_create_plist

Signature:

hid_t `H5Dget_create_plist`(*hid_t* `dataset_id`)

Purpose:

Returns an identifier for a copy of the dataset creation property list for a dataset.

Description:

`H5Dget_create_plist` returns an identifier for a copy of the dataset creation property list associated with the dataset specified by `dataset_id`.

The creation property list identifier should be released with `H5Pclose`.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to query.

Returns:

Returns a dataset creation property list identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5dget_create_plist_f

```
SUBROUTINE h5dget_create_plist_f(dataset_id, creation_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dataset_id    ! Dataset identifier
  INTEGER(HID_T), INTENT(OUT) :: creation_id  ! Dataset creation
                                              ! property list identifier
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure
END SUBROUTINE h5dget_create_plist_f
```

Name: H5Dget_offset

Signature:

haddr_t `H5Dget_offset`(*hid_t* `dset_id`)

Purpose:

Returns dataset address in file.

Description:

`H5Dget_offset` returns the address in the file of the dataset `dset_id`. That address is expressed as the offset in bytes from the beginning of the file.

Parameters:

hid_t `dset_id`	Dataset identifier.

Returns:

Returns the offset in bytes; otherwise returns `HADDR_UNDEF`, a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Dget_space

Signature:

hid_t `H5Dget_space`(*hid_t* `dataset_id`)

Purpose:

Returns an identifier for a copy of the dataspace for a dataset.

Description:

`H5Dget_space` returns an identifier for a copy of the dataspace for a dataset. The dataspace identifier should be released with the `H5Sclose` function.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to query.

Returns:

Returns a dataspace identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5dget_space_f

```
SUBROUTINE h5dget_space_f(dataset_id, dataspace_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dataset_id      ! Dataset identifier
  INTEGER(HID_T), INTENT(OUT) :: dataspace_id   ! Dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr                ! Error code
                                                ! 0 on success and -1 on failure
END SUBROUTINE h5dget_space_f
```

Name: H5Dget_space_status

Signature:

herr_t `H5Dget_space_status`(*hid_t* `dset_id`, *H5D_space_status_t* *`status`)

Purpose:

Determines whether space has been allocated for a dataset.

Description:

`H5Dget_space_status` determines whether space has been allocated for the dataset `dset_id`.

Space allocation status is returned in `status`, which will have one of the following values:

`H5D_SPACE_STATUS_NOT_ALLOCATED`	Space has not been allocated for this dataset.
`H5D_SPACE_STATUS_ALLOCATED`	Space has been allocated for this dataset.
`H5D_SPACE_STATUS_PART_ALLOCATED`	Space has been partially allocated for this dataset. (Used only for datasets with chunked storage.)

Parameters:

hid_t `dset_id`	IN: Identifier of the dataset to query.
H5D_space_status_t *`status`	OUT: Space allocation status.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5dget_space_status_f

```
SUBROUTINE h5dget_space_status_f(dset_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id  ! Dataset identifier
  INTEGER, INTENET(OUT)      :: flag     ! Status flag ; possible values:
                                         ! H5D_SPACE_STS_ERROR_F
                                         ! H5D_SPACE_STS_NOT_ALLOCATED_F
                                         ! H5D_SPACE_STS_PART_ALLOCATED_F
                                         ! H5D_SPACE_STS_ALLOCATED_F
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5dget_space_status_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Dget_storage_size

Signature:

hsize_t `H5Dget_storage_size`(*hid_t* `dataset_id`)

Purpose:

Returns the amount of storage required for a dataset.

Description:

`H5Dget_storage_size` returns the amount of storage that is required for the specified dataset, `dataset_id`. For chunked datasets, this is the number of allocated chunks times the chunk size. The return value may be zero if no data has been stored.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to query.

Returns:

Returns the amount of storage space allocated for the dataset, not counting meta data; otherwise returns 0 (zero).

Fortran90 Interface: h5dget_storage_size_f

```
SUBROUTINE h5dget_storage_size_f(dset_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id  ! Dataset identifier
  INTEGER(HSIZE_T), INTENT(OUT)  :: size ! Amount of storage required
                                         ! for dataset
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5dget_storage_size_f
```

History:

Release	**Fortran90**
1.4.5	Function introduced in this release.

Name: H5Dget_type

Signature:

hid_t `H5Dget_type`(*hid_t* `dataset_id`)

Purpose:

Returns an identifier for a copy of the datatype for a dataset.

Description:

`H5Dget_type` returns an identifier for a copy of the datatype for a dataset. The datatype should be released with the `H5Tclose` function.

If a dataset has a named datatype, then an identifier to the opened datatype is returned. Otherwise, the returned datatype is read-only. If atomization of the datatype fails, then the datatype is closed.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to query.

Returns:

Returns a datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5dget_type_f

```
SUBROUTINE h5dget_type_f(dataset_id, datatype_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dataset_id    ! Dataset identifier
  INTEGER(HID_T), INTENT(OUT) :: datatype_id  ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure
END SUBROUTINE h5dget_type_f
```

Name: H5Diterate

Signature:

herr_t `H5Diterate`(*void* *`buf`, *hid_t* `type_id`, *hid_t* `space_id`, *H5D_operator_t* `operator`, *void* *`operator_data`)

Purpose:

Iterates over all selected elements in a dataspace.

Description:

`H5Diterate` iterates over all the data elements in the memory buffer `buf`, executing the callback function `operator` once for each such data element.

The protoype of the callback function `operator` is as follows (as defined in the source code file `H5Lpublic.h`):

```
herr_t (*H5D_operator_t)(void elem, hid_t type_id, unsigned ndim,
                         const hsize_t *point, void *operator_data)
```

The parameters of this callback function have the following values or meanings:

void *`elem`	IN/OUT: Pointer to the memory buffer containing the current data element
hid_t `type_id`	IN: Datatype identifier for the elements stored in `elem`
unsigned `ndim`	IN: Number of dimensions for the `point` array
const hsize_t *`point`	IN: Array containing the location of the element within the original dataspace
void *`operator_data`	IN/OUT: Pointer to any user-defined data associated with the operation

The possible return values from the callback function, and the effect of each, are as follows:

◊ Zero causes the iterator to continue, returning zero when all data elements have been processed.
◊ A positive value causes the iterator to immediately return that positive value, indicating short-circuit success.
◊ A negative value causes the iterator to immediately return that value, indicating failure.

The `H5Diterate` `operator_data` parameter is a user-defined pointer to the data required to process dataset elements in the course of the iteration. If `operator` needs to pass data back to the application, such data can be returned in this same buffer. This pointer is passed back to each step of the iteration in the `operator` callback function's `operator_data` parameter.

Unlike other HDF5 iterators, this iteration operation cannot be restarted at the point of exit; a second `H5Diterate` call will always restart at the beginning.

Parameters:

void *`buf`	IN/OUT: Pointer to the buffer in memory containing the elements to iterate over
hid_t `type_id`	IN: Datatype identifier for the elements stored in `buf`
hid_t `space_id`	IN: Dataspace identifier for `buf`
H5D_operator_t `operator`	IN: Function pointer to the routine to be called for each element in `buf` iterated over
void *`operator_data`	IN/OUT: Pointer to any user-defined data associated with the operation

Returns:

Returns the return value of the last operator if it was non-zero, or zero if all elements have been processed. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.4	The following changes occured in the `H5D_operator_t` function in this release: `ndim` parameter type was changed to *unsigned* `point` parameter type was changed to *const hsize_t*

Name: H5Dopen

Signature:

hid_t `H5Dopen`(*hid_t* `loc_id`, *const char* `*name`)

hid_t `H5Dopen`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `dapl_id`)

Purpose:

Opens an existing dataset.

Description:

`H5Dopen` is a macro that is mapped to either `H5Dopen1` or `H5Dopen2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Dopen` is mapped to the most recent version of the function, currently `H5Dopen2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Dopen` will be mapped to `H5Dopen1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Dopen` mapping**
Global settings	
No compatibility flag	`H5Dopen2`
Enable deprecated symbols	`H5Dopen2`
Disable deprecated symbols	`H5Dopen2`
Emulate Release 1.6 interface	`H5Dopen1`
Function-level macros	
`H5Dopen_vers = 2`	`H5Dopen2`
`H5Dopen_vers = 1`	`H5Dopen1`

Fortran90 Interface: h5dopen_f

```
SUBROUTINE h5dopen_f(loc_id, name, dset_id, hdferr, dapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the dataset
  INTEGER(HID_T), INTENT(OUT) :: dset_id ! Dataset identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code:
                                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: dapl_id
                                         ! Dataset access property list
END SUBROUTINE h5dopen_f
```

History:

Release	**C**
1.8.0	The function `H5Dopen` renamed to `H5Dopen1` and deprecated in this release. The macro `H5Dopen` and the function `H5Dopen2` introduced in this release.

Name: H5Dopen1

Signature:

hid_t `H5Dopen1`(*hid_t* `loc_id`, *const char* *`name`)

Purpose:

Opens an existing dataset.

Notice:

This function is deprecated in favor of the function `H5Dopen2` *or the macro* `H5Dopen`*.*

Description:

`H5Dopen1` opens an existing dataset for access in the file or group specified in `loc_id`. `name` is a dataset name and is used to identify the dataset in the file.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group within which the dataset to be accessed will be found.
const char * `name`	IN: The name of the dataset to access.

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Dopen`.

History:

Release	**C**
1.8.0	Function `H5Dopen` renamed to `H5Dopen1` and deprecated in this release.

Name: H5Dopen2

Signature:

hid_t `H5Dopen2`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `dapl_id`)

Purpose:

Opens an existing dataset.

Description:

`H5Dopen2` opens the existing dataset specified by a location identifier and name, `loc_id` and `name`, respectively.

The dataset access property list, `dapl_id`, provides information regarding access to the dataset.

To conserve and release resources, the dataset should be closed when access is no longer required.

Parameters:

hid_t `loc_id`	IN: Location identifier
const char `*name`	IN: Dataset name
hid_t `dapl_id`	IN: Dataset access property list

Returns:

Returns a dataset identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Dopen`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Dread
Signature:

herr_t `H5Dread`(*hid_t* `dataset_id`, *hid_t* `mem_type_id`, *hid_t* `mem_space_id`, *hid_t* `file_space_id`, *hid_t* `xfer_plist_id`, *void* * `buf`)

Purpose:

Reads raw data from a dataset into a buffer.

Description:

`H5Dread` reads a (partial) dataset, specified by its identifier `dataset_id`, from the file into an application memory buffer `buf`. Data transfer properties are defined by the argument `xfer_plist_id`. The memory datatype of the (partial) dataset is identified by the identifier `mem_type_id`. The part of the dataset to read is defined by `mem_space_id` and `file_space_id`.

`file_space_id` is used to specify only the selection within the file dataset's dataspace. Any dataspace specified in `file_space_id` is ignored by the library and the dataset's dataspace is always used. `file_space_id` can be the constant `H5S_ALL`. which indicates that the entire file dataspace, as defined by the current dimensions of the dataset, is to be selected.

`mem_space_id` is used to specify both the memory dataspace and the selection within that dataspace. `mem_space_id` can be the constant `H5S_ALL`, in which case the file dataspace is used for the memory dataspace and the selection defined with `file_space_id` is used for the selection within that dataspace.

If raw data storage space has not been allocated for the dataset and a fill value has been defined, the returned buffer `buf` is filled with the fill value.

The behavior of the library for the various combinations of valid dataspace identifiers and H5S_ALL for the `mem_space_id` and the `file_space_id` parameters is described below:

`mem_space_id`	**`file_space_id`**	**Behavior**
valid dataspace identifier	valid dataspace identifier	`mem_space_id` specifies the memory dataspace and the selection within it. `file_space_id` specifies the selection within the file dataset's dataspace.
`H5S_ALL`	valid dataspace identifier	The file dataset's dataspace is used for the memory dataspace and the selection specified with `file_space_id` specifies the selection within it. The combination of the file dataset's dataspace and the selection from `file_space_id` is used for memory also.
valid dataspace identifier	`H5S_ALL`	`mem_space_id` specifies the memory dataspace and the selection within it. The selection within the file dataset's dataspace is set to the "all" selection.
`H5S_ALL`	`H5S_ALL`	The file dataset's dataspace is used for the memory dataspace and the selection within the memory dataspace is set to the "all" selection. The selection within the file dataset's dataspace is set to the "all" selection.

Setting an `H5S_ALL` selection indicates that the entire dataspace, as defined by the current dimensions of a dataspace, will be selected. The number of elements selected in the memory dataspace must match the number of elements selected in the file dataspace.

`xfer_plist_id` can be the constant `H5P_DEFAULT`. in which case the default data transfer properties are used.

Data is automatically converted from the file datatype and dataspace to the memory datatype and dataspace at the time of the read. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset read from.
hid_t `mem_type_id`	IN: Identifier of the memory datatype.
hid_t `mem_space_id`	IN: Identifier of the memory dataspace.
hid_t `file_space_id`	IN: Identifier of the dataset's dataspace in the file.
hid_t `xfer_plist_id`	IN: Identifier of a transfer property list for this I/O operation.
void * `buf`	OUT: Buffer to receive data read from file.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5dread_f, h5dread_vl_f

There is no direct FORTRAN couterpart for the C function `H5Dread`. Instead, that functionality is provided by two FORTRAN functions:

`h5dread_f`	**Purpose:** Reads data other than variable-length data.
`h5dread_vl_f`	**Purpose:** Reads variable-length data.

```
SUBROUTINE h5dread_f(dset_id, mem_type_id, buf, dims, hdferr, &
                     mem_space_id, file_space_id, xfer_prp)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id     ! Dataset identifier
  INTEGER(HID_T), INTENT(IN) :: mem_type_id ! Memory datatype identifier
  TYPE, INTENT(INOUT) :: buf                ! Data buffer; may be a scalar
                                            ! or an array
  DIMENSION(*), INTEGER(HSIZE_T), INTENT(IN)  :: dims
                                            ! Array to hold corresponding
                                            ! dimension sizes of data
                                            ! buffer buf
                                            ! dim(k) has value of the k-th
                                            ! dimension of buffer buf
                                            ! Values are ignored if buf is
                                            ! a scalar
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: mem_space_id
                                            ! Memory dataspace identfier
                                            ! Default value is H5S_ALL_F
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: file_space_id
                                            ! File dataspace identfier
                                            ! Default value is H5S_ALL_F
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: xfer_prp
                                            ! Transfer property list identifier
                                            ! Default value is H5P_DEFAULT_F
END SUBROUTINE h5dread_f
```

```
SUBROUTINE h5dread_vl_f(dset_id, mem_type_id, buf, dims, len, hdferr, &
                     mem_space_id, file_space_id, xfer_prp)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id     ! Dataset identifier
  INTEGER(HID_T), INTENT(IN) :: mem_type_id ! Memory datatype identifier
  TYPE, INTENT(INOUT), & DIMENSION(dims(1),dims(2)) :: buf
                                            ! Data buffer; may be a scalar
                                            ! or an array
                                            ! TYPE must be one of the following
                                            !     INTEGER
                                            !     REAL
                                            !     CHARACTER

  INTEGER(HSIZE_T), INTENT(IN), DIMENSION(2)  :: dims
                                            ! Array to hold corresponding
                                            ! dimension sizes of data
                                            ! buffer buf
                                            ! dim(k) has value of the k-th
                                            ! dimension of buffer buf
                                            ! Values are ignored if buf is
                                            ! a scalar
  INTEGER(SIZE_T), INTENT(INOUT), DIMENSION(*)  :: len
                                            ! Array to store length of
                                            ! each element
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: mem_space_id
                                            ! Memory dataspace identfier
                                            ! Default value is H5S_ALL_F
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: file_space_id
                                            ! File dataspace identfier
                                            ! Default value is H5S_ALL_F
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: xfer_prp
                                            ! Transfer property list identifier
                                            ! Default value is H5P_DEFAULT_F
END SUBROUTINE h5dread_vl_f
```

History:

Release	**Fortran90**
1.4.2	The `dims` parameter was added in this release.

Last modified: 15 May 2009

Name: H5Dset_extent

Signature:

herr_t `H5Dset_extent`(*hid_t* `dset_id`, *const hsize_t* `*size`)

Purpose:

Changes the sizes of a dataset's dimensions.

Description:

`H5Dset_extent` sets the current dimensions of the chunked dataset `dset_id` to the sizes specified in `size`.

`size` is a 1-dimensional array with *n* elements, where *n* is the rank of the dataset's current dataspace.

This function can be applied to the following datasets:

- ◊ A chunked dataset with unlimited dimensions
- ◊ A chunked dataset with fixed dimensions if the new dimension sizes are less than the maximum sizes set with `maxdims` (see `H5Screate_simple`)
- ◊ An external dataset with unlimited dimensions
- ◊ An external dataset with fixed dimensions if the new dimension sizes are less than the maximum sizes set with `maxdims`

  Note that external datasets are always contiguous and can be extended only along the first dimension.

Space on disk is immediately allocated for the new dataset extent if the dataset's space allocation time is set to `H5D_ALLOC_TIME_EARLY`. Fill values will be written to the dataset if the dataset's fill time is set to `H5D_FILL_TIME_IFSET` or `H5D_FILL_TIME_ALLOC`. (See `H5Pset_fill_time` and `H5Pset_alloc_time`.)

Note:

If the sizes specified in `size` are smaller than the dataset's current dimension sizes, `H5Dset_extent` will reduce the dataset's dimension sizes to the specified values. *It is the user's responsibility to ensure that valuable data is not lost;* `H5Dset_extent` *does not check.*

If it is necessary to ensure that current dimension sizes are not reduced, the function `H5Dextend` can be used.

Except for external datasets, `H5Dset_extent` is for use with chunked datasets only, not contiguous datasets.

Parameters:

hid_t `dset_id`	IN: Dataset identifier
const hsize_t `*size`	IN: Array containing the new magnitude of each dimension of the dataset.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: H5Dset_extent

```
SUBROUTINE h5dset_extent_f(dataset_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dataset_id   ! Dataset identifier
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN) :: size
                                             ! Array containing
                                             ! dimensions' sizes
  INTEGER, INTENT(OUT) :: hdferr             ! Error code:
```

```
                                              ! 0 on success and -1 on failure
      END SUBROUTINE h5dset_extent_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Dvlen_get_buf_size

Signature:

herr_t `H5Dvlen_get_buf_size`(*hid_t* `dataset_id`, *hid_t* `type_id`, *hid_t* `space_id`, *hsize_t* `*size`)

Purpose:

Determines the number of bytes required to store VL data.

Description:

`H5Dvlen_get_buf_size` determines the number of bytes required to store the VL data from the dataset, using the `space_id` for the selection in the dataset on disk and the `type_id` for the memory representation of the VL data in memory.

`*size` is returned with the number of bytes required to store the VL data in memory.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to query.
hid_t `type_id`	IN: Datatype identifier.
hid_t `space_id`	IN: Dataspace identifier.
hsize_t `*size`	OUT: The size in bytes of the memory buffer required to store the VL data.

Returns:

Returns non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5dvlen_get_max_len_f

There is no direct FORTRAN couterpart for the C function `H5Dvlen_get_buf_size`; corresponding functionality is provided by the FORTRAN function `h5dvlen_get_max_len_f`.

```
SUBROUTINE h5dvlen_get_max_len_f(dset_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id     ! Dataset identifier
  INTEGER(HID_T), INTENT(IN) :: type_id     ! Datatype identifier
  INTEGER(HID_T), INTENT(IN) :: space_id    ! Dataspace identifier

  INTEGER(SIZE_T), INTENT(OUT)  :: elem_len ! Maximum length of the element
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5dvlen_get_max_len_f
```

History:

Release	**C**	**Fortran90**
1.4.5		Function introduced in this release.
1.4.0	Function introduced in this release.	

Name: H5Dvlen_reclaim

Signature:

herr_t `H5Dvlen_reclaim`(*hid_t* `type_id`, *hid_t* `space_id`, *hid_t* `plist_id`, *void* `*buf`)

Purpose:

Reclaims VL datatype memory buffers.

Description:

`H5Dvlen_reclaim` reclaims memory buffers created to store VL datatypes.

The `type_id` must be the datatype stored in the buffer. The `space_id` describes the selection for the memory buffer to free the VL datatypes within. The `plist_id` is the dataset transfer property list which was used for the I/O transfer to create the buffer. And `buf` is the pointer to the buffer to be reclaimed.

The VL structures (`hvl_t`) in the user's buffer are modified to zero out the VL information after the memory has been reclaimed.

If nested VL datatypes were used to create the buffer, this routine frees them *from the bottom up*, releasing all the memory without creating memory leaks.

Parameters:

hid_t `type_id`	IN: Identifier of the datatype.
hid_t `space_id`	IN: Identifier of the dataspace.
hid_t `plist_id`	IN: Identifier of the property list used to create the buffer.
void `*buf`	IN: Pointer to the buffer to be reclaimed.

Returns:

Returns non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

Name: H5Dwrite

Signature:

herr_t `H5Dwrite`(*hid_t* `dataset_id`, *hid_t* `mem_type_id`, *hid_t* `mem_space_id`, *hid_t* `file_space_id`, *hid_t* `xfer_plist_id`, *const void* * `buf`)

Purpose:

Writes raw data from a buffer to a dataset.

Description:

`H5Dwrite` writes a (partial) dataset, specified by its identifier `dataset_id`, from the application memory buffer `buf` into the file. Data transfer properties are defined by the argument `xfer_plist_id`. The memory datatype of the (partial) dataset is identified by the identifier `mem_type_id`. The part of the dataset to write is defined by `mem_space_id` and `file_space_id`.

`file_space_id` is used to specify only the selection within the file dataset's dataspace. Any dataspace specified in `file_space_id` is ignored by the library and the dataset's dataspace is always used. `file_space_id` can be the constant `H5S_ALL`. which indicates that the entire file dataspace, as defined by the current dimensions of the dataset, is to be selected.

`mem_space_id` is used to specify both the memory dataspace and the selection within that dataspace. `mem_space_id` can be the constant `H5S_ALL`, in which case the file dataspace is used for the memory dataspace and the selection defined with `file_space_id` is used for the selection within that dataspace.

The behavior of the library for the various combinations of valid dataspace IDs and H5S_ALL for the `mem_space_id` and the `file_space_id` parameters is described below:

`mem_space_id`	**`file_space_id`**	**Behavior**
valid dataspace identifier	valid dataspace identifier	`mem_space_id` specifies the memory dataspace and the selection within it. `file_space_id` specifies the selection within the file dataset's dataspace.
H5S_ALL	valid dataspace identifier	The file dataset's dataspace is used for the memory dataspace and the selection specified with `file_space_id` specifies the selection within it. The combination of the file dataset's dataspace and the selection from `file_space_id` is used for memory also.
valid dataspace identifier	H5S_ALL	`mem_space_id` specifies the memory dataspace and the selection within it. The selection within the file dataset's dataspace is set to the "all" selection.
H5S_ALL	H5S_ALL	The file dataset's dataspace is used for the memory dataspace and the selection within the memory dataspace is set to the "all" selection. The selection within the file dataset's dataspace is set to the "all" selection.

Setting an "all" selection indicates that the entire dataspace, as defined by the current dimensions of a dataspace, will be selected. The number of elements selected in the memory dataspace must match the number of elements selected in the file dataspace.

`xfer_plist_id` can be the constant `H5P_DEFAULT`. in which case the default data transfer properties are used.

Writing to an dataset will fail if the HDF5 file was not opened with write access permissions.

Data is automatically converted from the memory datatype and dataspace to the file datatype and dataspace at the time of the write. See the Data Conversion section of *The Data Type Interface (H5T)* in the *HDF5 User's Guide* for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

If the dataset's space allocation time is set to `H5D_ALLOC_TIME_LATE` or `H5D_ALLOC_TIME_INCR` and the space for the dataset has not yet been allocated, that space is allocated when the first raw data is written to the dataset. Unused space in the dataset will be written with fill values at the same time if the dataset's fill time is set to `H5D_FILL_TIME_IFSET` or `H5D_FILL_TIME_ALLOC`. (Also see H5Pset_fill_time and H5Pset_alloc_time.)

If a dataset's storage layout is 'compact', care must be taken when writing data to the dataset in parallel. A compact dataset's raw data is cached in memory and may be flushed to the file from any of the parallel processes, so parallel applications should always attempt to write identical data to the dataset from all processes.

Parameters:

hid_t `dataset_id`	IN: Identifier of the dataset to write to.
hid_t `mem_type_id`	IN: Identifier of the memory datatype.
hid_t `mem_space_id`	IN: Identifier of the memory dataspace.
hid_t `file_space_id`	IN: Identifier of the dataset's dataspace in the file.
hid_t `xfer_plist_id`	IN: Identifier of a transfer property list for this I/O operation.
const void * `buf`	IN: Buffer with data to be written to the file.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5dwrite_f, h5dwrite_vl_f

There is no direct FORTRAN couterpart for the C function `H5Dwrite`. Instead, that functionality is provided by two FORTRAN functions:

`h5dwrite_f`	**Purpose:** Writes data other than variable-length data.
`h5dwrite_vl_f`	**Purpose:** Writes variable-length data.

```
SUBROUTINE h5dwrite_f(dset_id, mem_type_id, buf, dims, hdferr, &
                      mem_space_id, file_space_id, xfer_prp)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id      ! Dataset identifier
  INTEGER(HID_T), INTENT(IN) :: mem_type_id  ! Memory datatype identifier
  TYPE, INTENT(IN) :: buf                    ! Data buffer; may be a scalar
                                             ! or an array
  DIMENSION(*), INTEGER(HSIZE_T), INTENT(IN)  :: dims
                                             ! Array to hold corresponding
                                             ! dimension sizes of data
                                             ! buffer buf; dim(k) has value
                                             ! of the k-th dimension of
                                             ! buffer buf; values are
                                             ! ignored if buf is a scalar
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
```

```
                                              ! 0 on success and -1 on failure

    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: mem_space_id
                                              ! Memory dataspace identfier
                                              ! Default value is H5S_ALL_F
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: file_space_id
                                              ! File dataspace identfier
                                              ! Default value is H5S_ALL_F

    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: xfer_prp
                                              ! Transfer property list
                                              ! identifier; default value
                                              ! is H5P_DEFAULT_F
  END SUBROUTINE h5dwrite_f



  SUBROUTINE h5dwrite_vl_f(dset_id, mem_type_id, buf, dims, len, hdferr, &
                       mem_space_id, file_space_id, xfer_prp)
    IMPLICIT NONE
    INTEGER(HID_T), INTENT(IN) :: dset_id     ! Dataset identifier
    INTEGER(HID_T), INTENT(IN) :: mem_type_id ! Memory datatype identifier
    TYPE, INTENT(IN), & DIMENSION(dims(1),dims(2)) :: buf
                                             ! Data buffer; may be a scalar
                                             ! or an array
                                             ! TYPE must be one of the following
                                             !     INTEGER
                                             !     REAL
                                             !     CHARACTER
    INTEGER(HSIZE_T), INTENT(IN), DIMENSION(2)  :: dims
                                             ! Array to hold corresponding
                                             ! dimension sizes of data
                                             ! buffer buf
                                             ! dim(k) has value of the k-th
                                             ! dimension of buffer buf
                                             ! Values are ignored if buf is
                                             ! a scalar
    INTEGER(SIZE_T), INTENT(IN), DIMENSION(*)  :: len
                                             ! Array to store length of
                                             ! each element
    INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                             ! 0 on success and -1 on failure
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: mem_space_id
                                             ! Memory dataspace identfier
                                             ! Default value is H5S_ALL_F
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: file_space_id
                                             ! File dataspace identfier
                                             ! Default value is H5S_ALL_F
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: xfer_prp
                                             ! Transfer property list identifier
                                             ! Default value is H5P_DEFAULT_F
  END SUBROUTINE h5dwrite_vl_f
```

History:

Release	**Fortran90**
1.4.2	A `dims` parameter has been added.

H5E: Error Interface

Error API Functions

These functions provide error handling capabilities in the HDF5 environment.
In the following lists, italic type indicates a configurable macro.

The C Interfaces:

- *H5Eclear*
- H5Eclear1 *
- H5Eclear2
- H5Ecreate_stack
- H5Eclose_stack
- *H5Eprint*
- H5Eprint1 *
- H5Eprint2
- *H5Epush*
- H5Epush1 *
- H5Epush2
- H5Epop
- H5Eget_num
- H5Eget_major *
- H5Eget_minor *
- H5Eget_msg
- H5Ecreate_msg
- H5Eclose_msg
- H5Eregister_class
- H5Eunregister_class
- H5Eget_class_name
- H5Eauto_is_v2
- *H5Eset_auto*
- H5Eset_auto1 *
- H5Eset_auto2
- *H5Eget_auto*
- H5Eget_auto1 *
- H5Eget_auto2
- *H5Ewalk*
- H5Ewalk1 *
- H5Ewalk2
- H5Eget_current_stack
- H5Eset_current_stack

** Use of these functions is deprecated in Release 1.8.0.*

Alphabetical Listing

- H5Eauto_is_v2
- *H5Eclear*
- H5Eclear1 *
- H5Eclear2
- H5Eclose_msg
- H5Eclose_stack
- H5Ecreate_msg
- H5Ecreate_stack
- *H5Eget_auto*
- H5Eget_auto1 *
- H5Eget_auto2
- H5Eget_class_name
- H5Eget_current_stack
- H5Eget_major *
- H5Eget_minor *
- H5Eget_msg
- H5Eget_num
- H5Epop
- *H5Eprint*
- H5Eprint1 *
- H5Eprint2
- *H5Epush*
- H5Epush1 *
- H5Epush2
- H5Eregister_class
- *H5Eset_auto*
- H5Eset_auto1 *
- H5Eset_auto2
- H5Eset_current_stack
- H5Eunregister_class
- *H5Ewalk*
- H5Ewalk1 *
- H5Ewalk2

** Use of these functions is deprecated in Release 1.8.0.*

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5eclear_f
- h5eprint_f
- h5eset_auto_f
- h5eget_major_f
- h5eget_minor_f

The Error interface provides error handling in the form of a stack. The `FUNC_ENTER()` macro clears the error stack whenever an interface function is entered. When an error is detected, an entry is pushed onto the stack. As the functions unwind, additional entries are pushed onto the stack. The API function will return some indication that an error occurred and the application can print the error stack.

Certain API functions in the H5E package, such as `H5Eprint1`, do not clear the error stack. Otherwise, any function which does not have an underscore immediately after the package name will clear the error stack. For instance, `H5Fopen` clears the error stack while `H5F_open` does not.

An error stack has a fixed maximum size. If this size is exceeded then the stack will be truncated and only the inner-most functions will have entries on the stack. This is expected to be a rare condition.

Each thread has its own error stack, but since multi-threading has not been added to the library yet, this package maintains a single error stack. The error stack is statically allocated to reduce the complexity of handling errors within the H5E package.

Name: H5Eauto_is_v2

Signature:

herr_t `H5Eauto_is_v2`(*hid_t* `estack_id`, *unsigned* *`is_stack`)

Purpose:

Determines type of error stack.

Description:

`H5Eauto_is_v2` determines whether the error auto reporting function for an error stack conforms to the `H5E_auto2_t` typedef or the `H5E_auto1_t` typedef.

The `is_stack` parameter is set to `1` if the error stack conforms to `H5E_auto2_t` and `0` if it conforms to `H5E_auto1_t`.

Parameters:

hid_t `estack_id`	The error stack identifier
*unsigned** `is_stack`	A flag indicating which error stack typedef the specified error stack conforms to.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eclear

Signature:

herr_t `H5Eclear1`(`void`)

herr_t `H5Eclear2`(*hid_t* `estack_id`)

Purpose:

Clears an error stack.

Description:

`H5Eclear` is a macro that is mapped to either `H5Eclear1` or `H5Eclear2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Eclear` is mapped to the most recent version of the function, currently `H5Eclear2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Eclear` will be mapped to `H5Eclear1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Eclear` mapping**
Global settings	
No compatibility flag	`H5Eclear2`
Enable deprecated symbols	`H5Eclear2`
Disable deprecated symbols	`H5Eclear2`
Emulate Release 1.6 interface	`H5Eclear1`
Function-level macros	
`H5Eclear_vers = 2`	`H5Eclear2`
`H5Eclear_vers = 1`	`H5Eclear1`

Fortran90 Interface: h5eclear_f

```
SUBROUTINE h5eclear_f(hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(OUT) :: hdferr  ! Error code

END SUBROUTINE h5eclear_f
```

History:

Release	**C**
1.8.0	The function `H5Eclear` renamed to `H5Eclear1` and deprecated in this release. The macro `H5Eclear` and the function `H5Eclear2` introduced in this release.

Name: H5Eclear1

Signature:

herr_t `H5Eclear1`(`void`)

Purpose:

Clears the error stack for the current thread.

Notice:

This function is deprecated in favor of the function `H5Eclear2` *or the macro* `H5Eclear`.

Description:

`H5Eclear1` clears the error stack for the current thread.

The stack is also cleared whenever an API function is called, with certain exceptions (for instance, `H5Eprint1`).

Parameters:

None

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5eclear_f

See `H5Eclear`.

History:

Release	**C**
1.8.0	Function `H5Eclear` renamed to `H5Eclear1` and deprecated in this release.

Name: H5Eclear2

Signature:

herr_t `H5Eclear2`(*hid_t* `estack_id`)

Purpose:

Clears the specified error stack or the error stack for the current thread.

Description:

`H5Eclear2` clears the error stack specified by `estack_id`, or, if `estack_id` is set to `H5E_DEFAULT`, the error stack for the current thread.

`estack_id` is an error stack identifier, such as that returned by `H5Eget_current_stack`.

The current error stack is also cleared whenever an API function is called, with certain exceptions (for instance, `H5Eprint1` or `H5Eprint2`).

Parameters:

hid_t `estack_id`	IN: Error stack identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5eclear_f

See `H5Eclear`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eclose_msg

Signature:

herr_t `H5Eclose_msg`(*hid_t* `mesg_id`)

Purpose:

Closes an error message identifier.

Description:

`H5Eclose_msg` closes an error message identifier., which can be either a major or minor message.

Parameters:

hid_t `mesg_id`	IN: Error message identifier.

Returns:

Returns a non-negative value on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eclose_stack

Signature:

herr_t `H5Eclose_stack`(*hid_t* `estack_id`)

Purpose:

Closes object handle for error stack.

Description:

`H5Eclose_stack` closes the object handle for an error stack and releases its resources. `H5E_DEFAULT` cannot be closed.

Parameters:

hid_t `estack_id`	IN: Error stack identifier.

Returns:

Returns a non-negative value on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ecreate_msg

Signature:

hid_t `H5Ecreate_msg`(*hid_t* `class`, *H5E_type_t* `msg_type`, *const char** `mesg`)

Purpose:

Add major error message to an error class.

Description:

`H5Ecreate_msg` adds an error message to an error class defined by client library or application program. The error message can be either major or minor which is indicated by parameter `msg_type`.

Parameters:

hid_t `class`	IN: Error class identifier.
H5E_type_t `msg_type`	IN: The type of the error message. Valid values are `H5E_MAJOR` and `H5E_MINOR`.
*const char** `mesg`	IN: Major error message.

Returns:

Returns a message identifier on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ecreate_stack

Signature:

hid_t `H5Ecreate_stack`(*void*)

Purpose:

Creates a new empty error stack.

Description:

`H5Ecreate_stack` creates a new empty error stack and returns the new stack’s identifier.

Parameters:

None.

Returns:

Returns an error stack identifier on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eget_auto

Signature:

herr_t `H5Eget_auto(` *H5E_auto_t* * `func`, *void* **`client_data`)

herr_t `H5Eget_auto(` *hid_t* `estack_id`, *H5E_auto_t* * `func`, *void* **`client_data`)

Purpose:

Returns settings for automatic error stack traversal function and its data.

Description:

`H5Eget_auto` is a macro that is mapped to either `H5Eget_auto1` or `H5Eget_auto2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Eget_auto` is mapped to the most recent version of the function, currently `H5Eget_auto2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Eget_auto` will be mapped to `H5Eget_auto1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Eget_auto` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Eget_auto2`
Enable deprecated symbols	`H5Eget_auto2`
Disable deprecated symbols	`H5Eget_auto2`
Emulate Release 1.6 interface	`H5Eget_auto1`
<u>Function-level macros</u>	
`H5Eget_auto_vers = 2`	`H5Eget_auto2`
`H5Eget_auto_vers = 1`	`H5Eget_auto1`

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	The function `H5Eget_auto` renamed to `H5Eget_auto1` and deprecated in this release. The macro `H5Eget_auto` and the function `H5Eget_auto2` introduced in this release.

Name: H5Eget_auto1

Signature:

herr_t `H5Eget_auto1`(*H5E_auto1_t* * `func`, *void* **`client_data`)

Purpose:

Returns the current settings for the automatic error stack traversal function and its data.

Notice:

This function is deprecated in favor of the function `H5Eget_auto2` *or the macro* `H5Eget_auto`.

Description:

`H5Eget_auto1` returns the current settings for the automatic error stack traversal function, `func`, and its data, `client_data`. Either or both arguments may be null, in which case the value is not returned.

Parameters:

H5E_auto1_t * `func`	OUT: Current setting for the function to be called upon an error condition.
void **`client_data`	OUT: Current setting for the data passed to the error function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function `H5Eget_auto` renamed to `H5Eget_auto1` and deprecated in this release.

Name: H5Eget_auto2

Signature:

herr_t `H5Eget_auto2`(*hid_t* `estack_id`, *H5E_auto2_t* * `func`, *void* **`client_data`)

Purpose:

Returns the settings for the automatic error stack traversal function and its data.

Description:

`H5Eget_auto2` returns the settings for the automatic error stack traversal function, `func`, and its data, `client_data`, that are associated with the error stack specified by `estack_id`.

Either or both of the `func` and `client_data` arguments may be null, in which case the value is not returned.

Parameters:

hid_t `estack_id`	IN: Error stack identifier. `H5E_DEFAULT` indicates the current stack.
H5E_auto2_t * `func`	OUT: The function currently set to be called upon an error condition.
void **`client_data`	OUT: Data currently set to be passed to the error function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eget_class_name

Signature:

ssize_t `H5Eget_class_name`(*hid_t* `class_id`, *char** `name`, *size_t* `size`)

Purpose:

Retrieves error class name.

Description:

`H5Eget_class_name` retrieves the name of the error class specified by the class identifier. If non-NULL pointer is passed in for `name` and `size` is greater than zero, the class name of `size` long is returned. The length of the error class name is also returned. If NULL is passed in as name, only the length of class name is returned. If zero is returned, it means no name. User is responsible for allocated enough buffer for the name.

Parameters:

hid_t `class_id`	IN: Error class identifier.
*char** `name`	OUT: The name of the class to be queried.
size_t `size`	IN: The length of class name to be returned by this function.

Returns:

Returns non-negative value as on success; otherwise returns negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eget_current_stack

Signature:

hid_t `H5Eget_current_stack`(*void*)

Purpose:

Returns copy of current error stack.

Description:

`H5Eget_current_stack` copies the current error stack and returns an error stack identifier for the new copy.

Parameters:

None.

Returns:

Returns an error stack identifier on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eget_major
Signature:

*const char ** `H5Eget_major`(*H5E_major_t* `n`)

Purpose:

Returns a character string describing an error specified by a major error number.

Notice:

This function has been deprecated.

Description:

Given a major error number, `H5Eget_major` returns a constant character string that describes the error.

Parameters:

H5E_major_t `n`	IN: Major error number.

Returns:

Returns a character string describing the error if successful. Otherwise returns "Invalid major error number."

Fortran90 Interface: h5eget_major_f

```
SUBROUTINE h5eget_major_f(error_no, name, hdferr)
  INTEGER, INTENT(IN) :: error_no         !Major error number
  CHARACTER(LEN=*), INTENT(OUT) :: name   ! File name
  INTEGER, INTENT(OUT) :: hdferr          ! Error code

END SUBROUTINE h5eget_major_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Eget_minor
Signature:

const char * `H5Eget_minor`(*H5E_minor_t* `n`)

Purpose:

Returns a character string describing an error specified by a minor error number.

Notice:

This function has been deprecated.

Description:

Given a minor error number, `H5Eget_minor` returns a constant character string that describes the error.

Parameters:

H5E_minor_t `n`	IN: Minor error number.

Returns:

Returns a character string describing the error if successful. Otherwise returns "Invalid minor error number."

Fortran90 Interface: h5eget_minor_f

```
SUBROUTINE h5eget_minor_f(error_no, name, hdferr)
  INTEGER, INTENT(IN) :: error_no         !Major error number
  CHARACTER(LEN=*), INTENT(OUT) :: name   ! File name
  INTEGER, INTENT(OUT) :: hdferr          ! Error code

END SUBROUTINE h5eget_minor_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Eget_msg

Signature:

ssize_t `H5Eget_msg`(*hid_t* `mesg_id`, *H5E_type_t** `mesg_type`, *char** `mesg`, *size_t* `size`)

Purpose:

Retrieves an error message.

Description:

`H5Eget_msg` retrieves the error message including its length and type. The error message is specified by `mesg_id`. User is responsible for passing in enough buffer for the message. If `mesg` is not NULL and `size` is greater than zero, the error message of `size` long is returned. The length of the message is also returned. If NULL is passed in as `mesg`, only the length and type of the message is returned. If the return value is zero, it means no message.

Parameters:

hid_t `mesg_id`	IN: Idenfier for error message to be queried.
*H5E_type_t** `mesg_type`	OUT: The type of the error message. Valid values are `H5E_MAJOR` and `H5E_MINOR`.
*char** `mesg`	OUT: Error message buffer.
size_t `size`	IN: The length of error message to be returned by this function.

Returns:

Returns the size of the error message in bytes on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eget_num

Signature:

ssize_t `H5Eget_num`(*hid_t* `estack_id`)

Purpose:

Retrieves the number of error messages in an error stack.

Description:

`H5Eget_num` retrieves the number of error records in the error stack specified by `estack_id` (including major, minor messages and description).

Parameters:

hid_t `estack_id`	IN: Error stack identifier.

Returns:

Returns a non-negative value on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Epop

Signature:

herr_t `H5Epop`(*hid_t* `estack_id`, *size_t* `count`)

Purpose:

Deletes specified number of error messages from the error stack.

Description:

`H5Epop` deletes the number of error records specified in `count` from the top of the error stack specified by `estack_id` (including major, minor messages and description). The number of error messages to be deleted is specified by `count`.

Parameters:

hid_t `estack_id`	IN: Error stack identifier.
size_t `count`	IN: The number of error messages to be deleted from the top of error stack.

Returns:

Returns a non-negative value on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eprint

Signature:

herr_t `H5Eprint1`(*FILE* * `stream`)

herr_t `H5Eprint2`(*hid_t* `estack_id`, *FILE* * `stream`))

Purpose:

Prints an error stack in a default manner.

Description:

`H5Eprint` is a macro that is mapped to either `H5Eprint1` or `H5Eprint2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Eprint` is mapped to the most recent version of the function, currently `H5Eprint2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Eprint` will be mapped to `H5Eprint1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Eprint` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Eprint2`
Enable deprecated symbols	`H5Eprint2`
Disable deprecated symbols	`H5Eprint2`
Emulate Release 1.6 interface	`H5Eprint1`
<u>Function-level macros</u>	
`H5Eprint_vers = 2`	`H5Eprint2`
`H5Eprint_vers = 1`	`H5Eprint1`

Fortran90 Interface: h5eprint_f

```
SUBROUTINE h5eprint_f(hdferr, name)
  CHARACTER(LEN=*), OPTIONAL, INTENT(IN) :: name ! File name
  INTEGER, INTENT(OUT) :: hdferr                 ! Error code

END SUBROUTINE h5eprint_f
```

History:

Release	**C**
1.8.0	The function `H5Eprint` renamed to `H5Eprint1` and deprecated in this release. The macro `H5Eprint` and the function `H5Eprint2` introduced in this release.

Name: H5Eprint1

Signature:

herr_t `H5Eprint1`(*FILE* * `stream`)

Purpose:

Prints the current error stack in a default manner.

Notice:

This function is deprecated in favor of the function `H5Eprint2` or the macro `H5Eprint`.

Description:

`H5Eprint1` prints the error stack for the current thread on the specified stream, `stream`. Even if the error stack is empty, a one-line message will be printed:

```
  HDF5-DIAG: Error detected in thread 0.
```

`H5Eprint1` is a convenience function for `H5Ewalk1` with a function that prints error messages. Users are encouraged to write their own more specific error handlers.

Parameters:

FILE * `stream`	IN: File pointer, or stderr if NULL.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5eprint_f

See `H5Eprint`.

History:

Release	C
1.8.0	Function `H5Eprint` renamed to `H5Eprint1` and deprecated in this release.

Name: H5Eprint2

Signature:

herr_t `H5Eprint2`(*hid_t* `estack_id`, *FILE* * `stream`)

Purpose:

Prints the specified error stack in a default manner.

Description:

`H5Eprint2` prints the error stack specified by `estack_id` on the specified stream, `stream`. Even if the error stack is empty, a one-line message of the following form will be printed:

```
HDF5-DIAG: Error detected in HDF5 library version: 1.5.62 thread 0.
```

A similar line will appear before the error messages of each error class stating the library name, library version number, and thread identifier.

If `estack_id` is `H5E_DEFAULT`, the current error stack will be printed.

`H5Eprint2` is a convenience function for `H5Ewalk2` with a function that prints error messages. Users are encouraged to write their own more specific error handlers.

Parameters:

hid_t `estack_id`	IN: Identifier of the error stack to be printed. If the identifier is `H5E_DEFAULT`, the current error stack will be printed.
FILE * `stream`	IN: File pointer, or stderr if NULL.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Epush

Signature:

herr_t `H5Epush`(*const char* `*file`, *const char* `*func`, *unsigned* `line`, *H5E_major_t* `maj_num`, *H5E_minor_t* `min_num`, *const char* `*str`)

herr_t `H5Epush`(*hid_t* `estack_id`, *const char* `*file`, *const char* `*func`, *unsigned* `line`, *hid_t* `class_id`, *hid_t* `major_id`, *hid_t* `minor_id`, *const char* `*msg`, ...)

Purpose:

Pushes a new error message onto an error stack.

Description:

`H5Epush` is a macro that is mapped to either `H5Epush1` or `H5Epush2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Epush` is mapped to the most recent version of the function, currently `H5Epush2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Epush` will be mapped to `H5Epush1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Epush` mapping**
Global settings	
No compatibility flag	`H5Epush2`
Enable deprecated symbols	`H5Epush2`
Disable deprecated symbols	`H5Epush2`
Emulate Release 1.6 interface	`H5Epush1`
Function-level macros	
`H5Epush_vers = 2`	`H5Epush2`
`H5Epush_vers = 1`	`H5Epush1`

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	The function `H5Epush` renamed to `H5Epush1` and deprecated in this release. The macro `H5Epush` and the function `H5Epush2` introduced in this release.

Name: H5Epush1

Signature:

herr_t `H5Epush1`(*const char* *`file`, *const char* *`func`, *unsigned* `line`, *H5E_major_t* `maj_num`, *H5E_minor_t* `min_num`, *const char* *`str`)

Purpose:

Pushes new error record onto error stack.

Notice:

This function is deprecated in favor of the function `H5Epush2` or the macro `H5Epush`.

Description:

`H5Epush1` pushes a new error record onto the error stack for the current thread.

The error has major and minor numbers `maj_num` and `min_num`, the function `func` where the error was detected, the name of the file `file` where the error was detected, the line `line` within that file, and an error description string `str`.

The function name, filename, and error description strings must be statically allocated.

Parameters:

const char *`file`	IN: Name of the file in which the error was detected.
const char *`func`	IN: Name of the function in which the error was detected.
unsigned `line`	IN: Line within the file at which the error was detected.
H5E_major_t `maj_num`	IN: Major error number.
H5E_minor_t `min_num`	IN: Minor error number.
const char *`str`	IN: Error description string.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.4.0	Function introduced in this release.
1.8.0	Function `H5Epush` renamed to `H5Epush1` and deprecated in this release.

Name: H5Epush2

Signature:

herr_t `H5Epush2`(*hid_t* `estack_id`, *const char* *`file`, *const char* *`func`, *unsigned* `line`, *hid_t* `class_id`, *hid_t* `major_id`, *hid_t* `minor_id`, *const char* *`msg`, ...)

Purpose:

Pushes new error record onto error stack.

Description:

`H5Epush2` pushes a new error record onto the error stack specified by `estack_id`.

The error record contains the error class identifier `class_id`, the major and minor message identifiers `major_id` and `minor_id`, the function name `func` where the error was detected, the filename `file` and line number `line` within that file where the error was detected, and an error description `msg`.

The major and minor errors must be in the same error class.

The function name, filename, and error description strings must be statically allocated.

`msg` can be a format control string with additional arguments. This design of appending additional arguments is similar to the system and C functions `printf` and `fprintf`.

Parameters:

hid_t `estack_id`	IN: Identifier of the error stack to which the error record is to be pushed. If the identifier is `H5E_DEFAULT`, the error record will be pushed to the current stack.
const char *`file`	IN: Name of the file in which the error was detected.
const char *`func`	IN: Name of the function in which the error was detected.
unsigned `line`	IN: Line number within the file at which the error was detected.
hid_t `class_id`	IN: Error class identifier.
hid_t `major_id`	IN: Major error identifier.
hid_t `minor_id`	IN: Minor error identifier.
const char *`msg`	IN: Error description string.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eregister_class

Signature:

hid_t `H5Eregister_class`(*const char** `cls_name`, *const char** `lib_name`, *const char** `version`)

Purpose:

Registers a client library or application program to the HDF5 error API.

Description:

`H5Eregister_class` registers a client library or application program to the HDF5 error API so that the client library or application program can report errors together with HDF5 library. It receives an identifier for this error class for further error operations. The library name and version number will be printed out in the error message as preamble.

Parameters:

*const char** `cls_name`	IN: Name of the error class.
*const char** `lib_name`	IN: Name of the client library or application to which the error class belongs.
*const char** `version`	IN: Version of the client library or application to which the error class belongs. A NULL can be passed in.

Returns:

Returns a class identifier on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eset_auto
Signature:

herr_t `H5Eset_auto`(*H5E_auto_t* `func`, *void* *`client_data`)
herr_t `H5Eset_auto`(*hid_t* `estack_id`, *H5E_auto_t* `func`, *void* *`client_data`)

Purpose:

Returns settings for automatic error stack traversal function and its data.

Description:

`H5Eget_auto` is a macro that is mapped to either `H5Eset_auto1` or `H5Eset_auto2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Eset_auto` is mapped to the most recent version of the function, currently `H5Eset_auto2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Eset_auto` will be mapped to `H5Eset_auto1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Eset_auto` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Eset_auto2`
Enable deprecated symbols	`H5Eset_auto2`
Disable deprecated symbols	`H5Eset_auto2`
Emulate Release 1.6 interface	`H5Eset_auto1`
<u>Function-level macros</u>	
`H5Eset_auto_vers = 2`	`H5Eset_auto2`
`H5Eset_auto_vers = 1`	`H5Eset_auto1`

Fortran90 Interface: h5eset_auto_f

```
SUBROUTINE h5eset_auto_f(printflag, hdferr)
  INTEGER, INTENT(IN) :: printflag  !flag to turn automatic error
                                    !printing on or off
                                    !possible values are:
                                    !printon (1)
                                    !printoff(0)
  INTEGER, INTENT(OUT) :: hdferr    ! Error code

END SUBROUTINE h5eset_auto_f
```

History:

Release	**C**
1.8.0	The function `H5Eset_auto` renamed to `H5Eset_auto1` and deprecated in this release. The macro `H5Eset_auto` and the function `H5Eset_auto2` introduced in this release.

Name: H5Eset_auto1

Signature:

herr_t `H5Eset_auto1`(*H5E_auto1_t* `func`, *void* *`client_data`)

Purpose:

Turns automatic error printing on or off.

Description:

`H5Eset_auto1` turns on or off automatic printing of errors. When turned on (non-null `func` pointer), any API function which returns an error indication will first call `func`, passing it `client_data` as an argument.

When the library is first initialized the auto printing function is set to `H5Eprint1` (cast appropriately) and `client_data` is the standard error stream pointer, `stderr`.

Automatic stack traversal is always in the `H5E_WALK_DOWNWARD` direction.

Parameters:

H5E_auto1_t `func`	IN: Function to be called upon an error condition.
void *`client_data`	IN: Data passed to the error function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5eset_auto_f

See `H5Eset_auto`.

History:

Release	**C**
1.8.0	Function `H5Eset_auto` renamed to `H5Eset_auto1` and deprecated in this release.

Name: H5Eset_auto2

Signature:

herr_t `H5Eset_auto2`(*hid_t* `estack_id`, *H5E_auto2_t* `func`, *void* *`client_data`)

Purpose:

Turns automatic error printing on or off.

Description:

`H5Eset_auto2` turns on or off automatic printing of errors for the error stack specified with `estack_id`. An `estack_id` value of `H5E_DEFAULT` indicates the current stack.

When automatic printing is turned on, by the use of a non-null `func` pointer, any API function which returns an error indication will first call `func`, passing it `client_data` as an argument.

When the library is first initialized, the auto printing function is set to `H5Eprint2` (cast appropriately) and `client_data` is the standard error stream pointer, `stderr`.

Automatic stack traversal is always in the `H5E_WALK_DOWNWARD` direction.

Automatic error printing is turned off with a `H5Eset_auto2` call with a NULL `func` pointer.

Parameters:

hid_t `estack_id`	IN: Error stack identifier.
H5E_auto2_t `func`	IN: Function to be called upon an error condition.
void *`client_data`	IN: Data passed to the error function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5eset_auto_f

See `H5Eset_auto`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eset_current_stack

Signature:

herr_t `H5Eset_current_stack`(*hid_t* `estack_id`)

Purpose:

Replaces the current error stack.

Description:

`H5Eset_current_stack` replaces the content of the current error stack with a copy of the content of the error stack specified by `estack_id`.

Parameters:

hid_t `estack_id`	IN: Error stack identifier.

Returns:

Returns a non-negative value on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Eunregister_class

Signature:

herr_t `H5Eunregister_class`(*hid_t* `class_id`)

Purpose:

Removes an error class.

Description:

`H5Eunregister_class` removes the error class specified by `class_id`. All the major and minor errors in this class will also be closed.

Parameters:

hid_t `class_id`	IN: Error class identifier.

Returns:

Returns a non-negative value on success; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ewalk

Signature:

herr_t `H5Ewalk`(*H5E_direction_t* `direction`, *H5E_walk_t* `func`, *void* * `client_data`)

herr_t `H5Ewalk`(*hid_t* `estack_id`, *H5E_direction_t* `direction`, *H5E_walk_t* `func`, *void* * `client_data`)

Purpose:

Walks an error stack, calling a specified function.

Description:

`H5Ewalk` is a macro that is mapped to either `H5Ewalk1` or `H5Ewalk2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Ewalk` is mapped to the most recent version of the function, currently `H5Ewalk2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Ewalk` will be mapped to `H5Ewalk1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Ewalk` mapping**
Global settings	
No compatibility flag	`H5Ewalk2`
Enable deprecated symbols	`H5Ewalk2`
Disable deprecated symbols	`H5Ewalk2`
Emulate Release 1.6 interface	`H5Ewalk1`
Function-level macros	
`H5Ewalk_vers = 2`	`H5Ewalk2`
`H5Ewalk_vers = 1`	`H5Ewalk1`

Fortran90 Interface: h5ewalk_f

None.

History:

Release	**C**
1.8.0	The function `H5Ewalk` renamed to `H5Ewalk1` and deprecated in this release. The macro `H5Ewalk` and the function `H5Ewalk2` introduced in this release.

Name: H5Ewalk1

Signature:

herr_t `H5Ewalk1`(*H5E_direction_t* `direction`, *H5E_walk1_t* `func`, *void* * `client_data`)

Purpose:

Walks the error stack for the current thread, calling a specified function.

Notice:

This function is deprecated in favor of the function `H5Ewalk2` or the macro `H5Ewalk`.

Description:

`H5Ewalk1` walks the error stack for the current thread and calls the specified function for each error along the way.

`direction` determines whether the stack is walked from the inside out or the outside in. A value of `H5E_WALK_UPWARD` means begin with the most specific error and end at the API; a value of `H5E_WALK_DOWNWARD` means to start at the API and end at the inner-most function where the error was first detected.

`func` will be called for each error in the error stack. Its arguments will include an index number (beginning at zero regardless of stack traversal direction), an error stack entry, and the `client_data` pointer passed to `H5E_print`. The `H5E_walk1_t` prototype is as follows:

`typedef` *herr_t* `(*H5E_walk1_t)(`*int* `n,` *H5E_error1_t* *`err_desc,` *void* *`client_data)`

where the parameters have the following meanings:

int `n`

Indexed position of the error in the stack.

H5E_error1_t *`err_desc`

Pointer to a data structure describing the error. *(This structure is currently described only in the source code file* `hdf5/src/H5Epublic.h`*. That file also contains the definitive list of major and minor error codes. That information will eventually be presented as an appendix to this Reference Manual.)*

void *`client_data`

Pointer to client data in the format expected by the user-defined function.

Parameters:

H5E_direction_t `direction`	IN: Direction in which the error stack is to be walked.
H5E_walk1_t `func`	IN: Function to be called for each error encountered.
void * `client_data`	IN: Data to be passed with `func`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function `H5Ewalk` renamed to `H5Ewalk1` and deprecated in this release.

Name: H5Ewalk2

Signature:

herr_t `H5Ewalk2`(*hid_t* `estack_id`, *H5E_direction_t* `direction`, *H5E_walk2_t* `func`, *void* * `client_data`)

Purpose:

Walks the specified error stack, calling the specified function.

Description:

`H5Ewalk2` walks the error stack specified by `estack_id` for the current thread and calls the function specified in `func` for each error along the way.

If the value of `estack_id` is `H5E_DEFAULT`, then `H5Ewalk2` walks the current error stack.

`direction` specifies whether the stack is walked from the inside out or the outside in. A value of `H5E_WALK_UPWARD` means to begin with the most specific error and end at the API; a value of `H5E_WALK_DOWNWARD` means to start at the API and end at the innermost function where the error was first detected.

`func`, a function compliant with the `H5E_walk2_t` prototype, will be called for each error in the error stack. Its arguments will include an index number `n` (beginning at zero regardless of stack traversal direction), an error stack entry `err_desc`, and the `client_data` pointer passed to `H5E_print`. The `H5E_walk2_t` prototype is as follows:

`typedef` *herr_t* `(*H5E_walk2_t)(`*unsigned* `n,` *const H5E_error2_t* `*err_desc,` *void* `*client_data)`

where the parameters have the following meanings:

unsigned `n`
: Indexed position of the error in the stack.

const H5E_error2_t `*err_desc`
: Pointer to a data structure describing the error. *(This structure is currently described only in the source code file `hdf5/src/H5Epublic.h`. That file also contains the definitive list of major and minor error codes; that information will eventually be presented as an appendix to this HDF5 Reference Manual.)*

void `*client_data`
: Pointer to client data in the format expected by the user-defined function.

Parameters:

hid_t `estack_id`	IN: Error stack identifier.
H5E_direction_t `direction`	IN: Direction in which the error stack is to be walked.
H5E_walk2_t `func`	IN: Function to be called for each error encountered.
void * `client_data`	IN: Data to be passed with `func`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

H5F: File Interface

File API Functions

These functions are designed to provide file-level access to HDF5 files. Further manipulation of objects inside a file is performed through one of APIs documented below.

The C Interfaces:

- H5Fcreate
- H5Fopen
- H5Freopen
- H5Fclose
- H5Fflush
- H5Fis_hdf5
- H5Fmount
- H5Funmount
- H5Fget_vfd_handle
- H5Fget_filesize
- H5Fget_create_plist
- H5Fget_access_plist
- H5Fget_info
- H5Fget_intent
- H5Fget_name
- H5Fget_obj_count
- H5Fget_obj_ids
- H5Fget_freespace
- H5Fget_mdc_config
- H5Fget_mdc_hit_rate
- H5Fget_mdc_size
- H5Freset_mdc_hit_rate_stats
- H5Fset_mdc_config

Alphabetical Listing

- H5Fclose
- H5Fcreate
- H5Fflush
- H5Fget_access_plist
- H5Fget_create_plist
- H5Fget_filesize
- H5Fget_freespace
- H5Fget_info
- H5Fget_intent
- H5Fget_mdc_config
- H5Fget_mdc_hit_rate
- H5Fget_mdc_size
- H5Fget_name
- H5Fget_obj_count
- H5Fget_obj_ids
- H5Fget_vfd_handle
- H5Fis_hdf5
- H5Fmount
- H5Fopen
- H5Freopen
- H5Freset_mdc_hit_rate_stats
- H5Fset_mdc_config
- H5Funmount

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5fcreate_f
- h5fopen_f
- h5freopen_f
- h5fclose_f
- h5fflush_f
- h5fis_hdf5_f
- h5fmount_f
- h5funmount_f
- h5fget_vfd_handle_f
- h5fget_filesize_f
- h5fget_freespace_f
- h5fget_create_plist_f
- h5fget_access_plist_f
- h5fget_name_f
- h5fget_obj_count_f
- h5fget_obj_ids_f

Last modified: 9 April 2009

Name: H5Fclose

Signature:

herr_t `H5Fclose`(*hid_t* `file_id`)

Purpose:

Terminates access to an HDF5 file.

Description:

`H5Fclose` terminates access to an HDF5 file by flushing all data to storage and terminating access to the file through `file_id`.

If this is the last file identifier open for the file and no other access identifier is open (e.g., a dataset identifier, group identifier, or shared datatype identifier), the file will be fully closed and access will end.

Delayed close:
Note the following deviation from the above-described behavior. If `H5Fclose` is called for a file but one or more objects within the file remain open, those objects will remain accessible until they are individually closed. Thus, if the dataset `data_sample` is open when `H5Fclose` is called for the file containing it, `data_sample` will remain open and accessible (including writable) until it is explicitely closed. The file will be automatically closed once all objects in the file have been closed.

Be warned, however, that there are circumstances where it is not possible to delay closing a file. For example, an MPI-IO file close is a collective call; all of the processes that opened the file must close it collectively. The file cannot be closed at some time in the future by each process in an independent fashion. Another example is that an application using an AFS token-based file access privilage may destroy its AFS token after `H5Fclose` has returned successfully. This would make any future access to the file, or any object within it, illegal.

In such situations, applications must close all open objects in a file before calling `H5Fclose`. It is generally recommended to do so in all cases.

Parameters:

hid_t `file_id`	IN: Identifier of a file to terminate access to.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5fclose_f

```
SUBROUTINE h5fclose_f(file_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: file_id ! File identifier
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5fclose_f
```

Last modified: 15 May 2009

Name: H5Fcreate

Signature:

hid_t `H5Fcreate`(*const char* *`name`, *unsigned* `flags`, *hid_t* `fcpl_id`, *hid_t* `fapl_id`)

Purpose:

Creates an HDF5 file.

Description:

`H5Fcreate` is the primary function for creating HDF5 files; it creates a new HDF5 file with the specified name and property lists and specifies whether an existing file of same name should be overwritten.

The `name` parameter specifies the name of the new file.

The `flags` parameter specifies whether an existing file is to be overwritten. It should be set to either `H5F_ACC_TRUNC` to overwrite an existing file or `H5F_ACC_EXCL`, instructing the function to fail if the file already exists.

New files are always created in read-write mode, so the read-write and read-only flags, `H5F_ACC_RDWR` and `H5F_ACC_RDONLY`, respectively, are not relevant in this function. Further note that a specification of `H5F_ACC_RDONLY` will be ignored; the file will be created in read-write mode, regardless.

More complex behaviors of file creation and access are controlled through the file creation and file access property lists, `fcpl_id` and `fapl_id`, respectively. The value of `H5P_DEFAULT` for any property list value indicates that the library should use the default values for that appropriate property list.

The return value is a file identifier for the newly-created file; this file identifier should be closed by calling `H5Fclose` when it is no longer needed.

Special case -- File creation in the case of an already-open file:
If a file being created is already opened, by either a previous `H5Fopen` or `H5Fcreate` call, the HDF5 library may or may not detect that the open file and the new file are the same physical file. (See `H5Fopen` regarding the limitations in detecting the re-opening of an already-open file.)

If the library detects that the file is already opened, `H5Fcreate` will return a failure, regardless of the use of `H5F_ACC_TRUNC`.

If the library does not detect that the file is already opened and `H5F_ACC_TRUNC` is not used, `H5Fcreate` will return a failure because the file already exists. Note that this is correct behavior.

But if the library does not detect that the file is already opened and `H5F_ACC_TRUNC` is used, `H5Fcreate` will truncate the existing file and return a valid file identifier. Such a truncation of a currently-opened file will almost certainly result in errors. While unlikely, the HDF5 library may not be able to detect, and thus report, such errors.

Applications should avoid calling `H5Fcreate` with an already opened file.

Parameters:

const char *`name`	IN: Name of the file to access.
uintn `flags`	IN: File access flags. Allowable values are: `H5F_ACC_TRUNC`

Truncate file, if it already exists, erasing all data previously stored in the file.

`H5F_ACC_EXCL`

Fail if file already exists.

- `H5F_ACC_TRUNC` and `H5F_ACC_EXCL` are mutually exclusive; use exactly one.
- An additional flag, `H5F_ACC_DEBUG`, prints debug information. This flag can be combined with one of the above values using the bit-wise OR operator (`|'), but it is used only by HDF5 Library developers; *it is neither tested nor supported* for use in applications.

hid_t `fcpl_id`	IN: File creation property list identifier, used when modifying default file meta-data. Use `H5P_DEFAULT` to specify default file creation properties.
hid_t `fapl_id`	IN: File access property list identifier. If parallel file access is desired, this is a collective call according to the communicator stored in the `fapl_id`. Use `H5P_DEFAULT` for default file access properties.

Returns:

Returns a file identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5fcreate_f

```
SUBROUTINE h5fcreate_f(name, access_flags, file_id, hdferr, &
                        creation_prp, access_prp)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the file
  INTEGER, INTENT(IN) :: access_flag     ! File access flags
                                         ! Possible values are:
                                         !     H5F_ACC_RDWR_F
                                         !     H5F_ACC_RDONLY_F
                                         !     H5F_ACC_TRUNC_F
                                         !     H5F_ACC_EXCL_F
                                         !     H5F_ACC_DEBUG_F
  INTEGER(HID_T), INTENT(OUT) :: file_id ! File identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: creation_prp
                                         ! File creation propertly
                                         ! list identifier, if not
                                         ! specified its value is
                                         ! H5P_DEFAULT_F
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: access_prp
                                         ! File access property list
                                         ! identifier, if not
                                         ! specified its value is
                                         ! H5P_DEFAULT_F
END SUBROUTINE h5fcreate_f
```

Name: H5Fflush

Signature:

herr_t `H5Fflush`(*hid_t* `object_id`, *H5F_scope_t* `scope`)

Purpose:

Flushes all buffers associated with a file to disk.

Description:

`H5Fflush` causes all buffers associated with a file to be immediately flushed to disk without removing the data from the cache.

`object_id` can be any object associated with the file, including the file itself, a dataset, a group, an attribute, or a named data type.

`scope` specifies whether the scope of the flushing action is global or local. Valid values are

`H5F_SCOPE_GLOBAL`	Flushes the entire virtual file.
`H5F_SCOPE_LOCAL`	Flushes only the specified file.

Note:

HDF5 does not possess full control over buffering. `H5Fflush` flushes the internal HDF5 buffers then asks the operating system (the OS) to flush the system buffers for the open files. After that, the OS is responsible for ensuring that the data is actually flushed to disk.

Parameters:

hid_t `object_id`	IN: Identifier of object used to identify the file.
H5F_scope_t `scope`	IN: Specifies the scope of the flushing action.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5fflush_f

```
SUBROUTINE h5fflush_f(obj_id, scope, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: obj_id      ! Object identifier
  INTEGER, INTENT(IN)         :: scope       ! Flag with two possible values:
                                             !     H5F_SCOPE_GLOBAL_F
                                             !     H5F_SCOPE_LOCAL_F
  INTEGER, INTENT(OUT)        :: hdferr      ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5fflush_f
```

Name: H5Fget_access_plist

Signature:

hid_t `H5Fget_access_plist`(*hid_t* `file_id`)

Purpose:

Returns a file access property list identifier.

Description:

`H5Fget_access_plist` returns the file access property list identifier of the specified file.

See "File Access Properties" in H5P: Property List Interface in this reference manual and "File Access Property Lists" in *Files* in the *HDF5 User's Guide* for additional information and related functions.

Parameters:

hid_t `file_id`	IN: Identifier of file to get access property list of

Returns:

Returns a file access property list identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5fget_access_plist_f

```
SUBROUTINE h5fget_access_plist_f(file_id, fcpl_id, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: file_id ! File identifier
  INTEGER(HID_T), INTENT(OUT)  :: fapl_id ! File access property list identifier
  INTEGER, INTENT(OUT)         :: hdferr  ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5fget_access_plist_f
```

Name: H5Fget_create_plist

Signature:

hid_t `H5Fget_create_plist(`*hid_t* `file_id )`

Purpose:

Returns a file creation property list identifier.

Description:

`H5Fget_create_plist` returns a file creation property list identifier identifying the creation properties used to create this file. This function is useful for duplicating properties when creating another file.

See "File Creation Properties" in H5P: Property List Interface in this reference manual and "File Creation Properties" in *Files* in the *HDF5 User's Guide* for additional information and related functions.

The creation property list identifier should be released with `H5Pclose`.

Parameters:

hid_t `file_id`	IN: File identifier

Returns:

Returns a file creation property list identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5fget_create_plist_f

```
SUBROUTINE h5fget_create_plist_f(file_id, fcpl_id, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: file_id ! File identifier
  INTEGER(HID_T), INTENT(OUT)  :: fcpl_id ! File creation property list
                                          ! identifier
  INTEGER, INTENT(OUT)         :: hdferr  ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5fget_create_plist_f
```

Name: H5Fget_filesize

Signature:

herr_t `H5Fget_filesize`(*hid_t* `file_id`, *hsize_t* *`size`)

Purpose:

Returns the size of an HDF5 file.

Description:

`H5Fget_filesize` returns the size of the HDF5 file specified by `file_id`.

The returned size is that of the entire file, as opposed to only the HDF5 portion of the file. I.e., `size` includes the user block, if any, the HDF5 portion of the file, and any data that may have been appended beyond the data written through the HDF5 Library.

Parameters:

hid_t `file_id`

IN: Identifier of a currently-open HDF5 file

hsize_t *`size`

OUT: Size of the file, in bytes.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5fget_freespace_f

```
SUBROUTINE h5fget_filesize_f(file_id, size, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: file_id    ! file identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: size    ! Size of the file
  INTEGER, INTENT(OUT) :: hdferr           ! Error code: 0 on success,
                                           ! -1 if fail
END SUBROUTINE h5fget_filesize_f
```

History:

Release	**C**
1.6.3	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Fget_freespace

Signature:

hssize_t `H5Fget_freespace`(*hid_t* `file_id`)

Purpose:

Returns the amount of free space in a file.

Description:

Given the identifier of an open file, `file_id`, `H5Fget_freespace` returns the amount of space that is unused by any objects in the file.

Currently, the HDF5 library only tracks free space in a file from a file open or create until that file is closed, so this routine will only report the free space that has been created during that interval.

Parameters:

hid_t `file_id`	IN: Identifier of a currently-open HDF5 file

Returns:

Returns the amount of free space in the file if successful; otherwise returns a negative value.

Fortran90 Interface: h5fget_freespace_f

```
SUBROUTINE h5fget_freespace_f(file_id, free_space, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: file_id       ! File identifier
  INTEGER(HSSIZE_T), INTENT(OUT) :: free_space ! Amount of free space in file
  INTEGER, INTENT(OUT)        :: hdferr        ! Error code
                                               ! 0 on success and -1 on failure
END SUBROUTINE h5fget_freespace_f
```

History:

Release	**C**
1.6.1	Function introduced in this release.

Name: H5Fget_info

Signature:

herr_t `H5Fget_info`(*hid_t* `obj_id`, *H5F_info_t* `*file_info`)

Purpose:

Returns global information for a file.

Description:

`H5Fget_info` returns global information for the file associated with the object identifier `obj_id` in the *H5F_info_t* struct named `file_info`.

`obj_id` is an identifier for any object in the file of interest.

An *H5F_info_t* struct is defined as follows (in `H5Fpublic.h`):

```
        typedef struct H5F_info_t {
            hsize_t           super_ext_size;
            struct {
                hsize_t       hdr_size;
                H5_ih_info_t  msgs_info;
            } sohm;
        } H5F_info_t;
```

`super_ext_size` is the size of the superblock extension.

The *sohm* sub-struct contains shared object header message information: `hdr_size` is the size of shared of object header messages. `msgs_info` is a `H5_ih_info_t` struct containing the cumulative shared object header message index size and heap size; an *H5_ih_info_t* struct is defined as follows (in `H5public.h`):

```
        typedef struct H5_ih_info_t {
            hsize_t     index_size;
            hsize_t     heap_size;
        } H5_ih_info_t;
```

`index_size` is the summed size of all of the shared of object header indexes. Each index might be either a B-tree or a list. `heap_size` is the size of the heap.

Parameters:

hid_t `obj_id`,	IN: Object identifier for any object in the file.
H5F_info_t `*file_info`	OUT: Struct containing global file information.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Fget_intent

Signature:

herr_t `H5Fget_intent`(*hid_t* `file_id`, *unsigned* *`intent`)

Purpose:

Determines the read/write or read-only status of a file.

Description:

Given the identifier of an open file, `file_id`, `H5Fget_intent` retrieves the â’ intended access modeâ’ flag passed with `H5Fopen` when the file was opened.

The value of the flag is returned in `intent`. Valid values are as follows:

`H5F_ACC_RDWR`	File was opened with read/write access.
`H5F_ACC_RDONLY`	File was opened with read-only access.

The function will not return an error if `intent` is `NULL`; it will simply do nothing.

Parameters:

hid_t `file_id`	IN: File identifier for a currently-open HDF5 file
unsigned *`intent`	OUT: Intended access mode flag, as originally passed with `H5Fopen`.

Returns:

Returns the amount of free space in the file if successful; otherwise returns a negative value.

Fortran90 Interface: None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Fget_mdc_config

Signature:

herr_t `H5Fget_mdc_config`(*hid_t* `file_id`, *H5AC_cache_config_t* *`config_ptr`)

Purpose:

Obtain current metadata cache configuration for target file.

Description:

`H5Fget_mdc_config` loads the current metadata cache configuration into the instance of `H5AC_cache_config_t` pointed to by the `config_ptr` parameter.

Note that the version field of *`config_ptr` must be initialized --this allows the library to support old versions of the `H5AC_cache_config_t` structure.

See the overview of the metadata cache in the special topics section of the user manual for details on metadata cache configuration. If you haven't read and understood that documentation, the results of this call will not make much sense.

Parameters:

hid_t `file_id`	IN: Identifier of the target file
H5AC_cache_config_t *`config_ptr`	IN/OUT: Pointer to the instance of H5AC_cache_config_t in which the current metadata cache configuration is to be reported. The fields of this structure are discussed below:
General configuration section:	
int `version`	IN: Integer field indicating the the version of the H5AC_cache_config_t in use. This field should be set to H5AC__CURR_CACHE_CONFIG_VERSION (defined in H5ACpublic.h).
hbool_t `rpt_fcn_enabled`	OUT: Boolean flag indicating whether the adaptive cache resize report function is enabled. This field should almost always be set to FALSE. Since resize algorithm activity is reported via stdout, it MUST be set to FALSE on Windows machines. The report function is not supported code, and can be expected to change between versions of the library. Use it at your own risk.
hbool_t `open_trace_file`	OUT: Boolean field indicating whether the `trace_file_name` field should be used to open a trace file for the cache. This field will always be set to `FALSE` in this context.
hbool_t `close_trace_file`	OUT: Boolean field indicating whether the current trace file (if any) should be closed. This field will always be set to `FALSE` in this context.

char *`trace_file_name`	OUT: Full path name of the trace file to be opened if the `open_trace_file` field is `TRUE`. This field will always be set to the empty string in this context.
hbool_t `evictions_enabled`	OUT: Boolean flag indicating whether metadata cache entry evictions are enabled.
hbool_t `set_initial_size`	OUT: Boolean flag indicating whether the cache should be created with a user specified initial maximum size. If the configuration is loaded from the cache, this flag will always be FALSE.
size_t `initial_size`	OUT: Initial maximum size of the cache in bytes, if applicable. If the configuration is loaded from the cache, this field will contain the cache maximum size as of the time of the call.
double `min_clean_fraction`	OUT: Float value specifing the minimum fraction of the cache that must be kept either clean or empty when possible.
size_t `max_size`	OUT: Upper bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
size_t `min_size`	OUT: Lower bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
long int `epoch_length`	OUT: Number of cache accesses between runs of the adaptive cache resize code.
Increment configuration section:	
enum H5C_cache_incr_mode `incr_mode`	OUT: Enumerated value indicating the operational mode of the automatic cache size increase code. At present, only the following values are legal: H5C_incr__off: Automatic cache size increase is disabled. H5C_incr__threshold: Automatic cache size increase is enabled using the hit rate threshold algorithm.

double `lower_hr_threshold`	OUT: Hit rate threshold used in the hit rate threshold cache size increase algorithm.
double `increment`	OUT: The factor by which the current maximum cache size is multiplied to obtain an initial new maximum cache size if a size increase is triggered in the hit rate threshold cache size increase algorithm.
hbool_t `apply_max_increment`	OUT: Boolean flag indicating whether an upper limit will be applied to the size of cache size increases.
size_t `max_increment`	OUT: The maximum number of bytes by which the maximum cache size can be increased in a single step -- if applicable.
enum H5C_cache_flash_incr_mode `flash_incr_mode`	OUT: Enumerated value indicating the operational mode of the flash cache size increase code. At present, only the following values are legal: H5C_flash_incr__off: Flash cache size increase is disabled. H5C_flash_incr__add_space: Flash cache size increase is enabled using the add space algorithm.
double `flash_threshold`	OUT: The factor by which the current maximum cache size is multiplied to obtain the minimum size entry / entry size increase which may trigger a flash cache size increase.
double `flash_multiple`	OUT: The factor by which the size of the triggering entry / entry size increase is multiplied to obtain the initial cache size increment. This increment may be reduced to reflect existing free space in the cache and the `max_size` field above.

Decrement configuration section:

enum H5C_cache_decr_mode `decr_mode`	OUT: Enumerated value indicating the operational mode of the automatic cache size decrease code. At present, the following values are legal: H5C_decr__off: Automatic cache size decrease is disabled, and the remaining decrement fields are ignored. H5C_decr__threshold: Automatic cache size decrease is enabled using the hit rate threshold algorithm. H5C_decr__age_out: Automatic cache size decrease is enabled using the ageout algorithm. H5C_decr__age_out_with_threshold: Automatic cache size decrease is enabled using the ageout with hit rate threshold algorithm
double `upper_hr_threshold`	OUT: Upper hit rate threshold. This value is only used if the decr_mode is either H5C_decr__threshold or H5C_decr__age_out_with_threshold.
double `decrement`	OUT: Factor by which the current max cache size is multiplied to obtain an initial value for the new cache size when cache size reduction is triggered in the hit rate threshold cache size reduction algorithm.
hbool_t `apply_max_decrement`	OUT: Boolean flag indicating whether an upper limit should be applied to the size of cache size decreases.
size_t `max_decrement`	OUT: The maximum number of bytes by which cache size can be decreased if any single step, if applicable.
int `epochs_before_eviction`	OUT: The minimum number of epochs that an entry must reside unaccessed in cache before being evicted under either of the ageout cache size reduction algorithms.
hbool_t `apply_empty_reserve`	OUT: Boolean flag indicating whether an empty reserve should be maintained under either of the ageout cache size reduction algorithms.
double `empty_reserve`	OUT: Empty reserve for use with the ageout cache size reduction algorithms, if applicable.

Parallel configuration section:

int `dirty_bytes_threshold`	OUT: Threshold number of bytes of dirty metadata generation for triggering synchronizations of the metadata caches serving the target file in the parallel case. Synchronization occurs whenever the number of bytes of dirty metadata created since the last synchronization exceeds this limit.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Name: H5Fget_mdc_hit_rate

Signature:

herr_t `H5Fget_mdc_hit_rate`(*hid_t* `file_id`, *double* *`hit_rate_ptr`)

Purpose:

Obtain target file's metadata cache hit rate.

Description:

H5Fget_mdc_hit_rate queries the metadata cache of the target file to obtain its hit rate (cache hits / (cache hits + cache misses)) since the last time hit rate statistics were reset. If the cache has not been accessed since the last time the hit rate stats were reset, the hit rate is defined to be 0.0.

The hit rate stats can be reset either manually (via H5Freset_mdc_hit_rate_stats()), or automatically. If the cache's adaptive resize code is enabled, the hit rate stats will be reset once per epoch. If they are reset manually as well, the cache may behave oddly.

See the overview of the metadata cache in the special topics section of the user manual for details on the metadata cache and its adaptive resize algorithms.

Parameters:

hid_t `file_id`	IN: Identifier of the target file.
double * `hit_rate_ptr`	OUT: Pointer to the double in which the hit rate is returned. Note that *`hit_rate_ptr` is undefined if the API call fails.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Name: H5Fget_mdc_size

Signature:

herr_t `H5Fget_mdc_size`(*hid_t* `file_id`, *size_t* `*max_size_ptr`, *size_t* `*min_clean_size_ptr`, *size_t* `*cur_size_ptr`, *int* `*cur_num_entries_ptr`)

Purpose:

Obtain current metadata cache size data for specified file.

Description:

H5Fget_mdc_size queries the metadata cache of the target file for the desired size information, and returns this information in the locations indicated by the pointer parameters. If any pointer parameter is NULL, the associated data is not returned.

If the API call fails, the values returned via the pointer parameters are undefined.

If adaptive cache resizing is enabled, the cache maximum size and minimum clean size may change at the end of each epoch. Current size and current number of entries can change on each cache access.

Current size can exceed maximum size under certain conditions. See the overview of the metadata cache in the special topics section of the user manual for a discussion of this.

Parameters:

hid_t `file_id`	IN: Identifier of the target file.
size_t `*max_size_ptr`	OUT: Pointer to the location in which the current cache maximum size is to be returned, or NULL if this datum is not desired.
size_t `*min_clean_size_ptr`	OUT: Pointer to the location in which the current cache minimum clean size is to be returned, or NULL if that datum is not desired.
size_t `*cur_size_ptr`	OUT: Pointer to the location in which the current cache size is to be returned, or NULL if that datum is not desired.
int `*cur_num_entries_ptr`	OUT: Pointer to the location in which the current number of entries in the cache is to be returned, or NULL if that datum is not desired.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Name: H5Fget_name

Signature:

ssize_t `H5Fget_name`(*hid_t* `obj_id`, *char* *`name`, *size_t* `size`)

Purpose:

Retrieves name of file to which object belongs.

Description:

`H5Fget_name` retrieves the name of the file to which the object `obj_id` belongs. The object can be a group, dataset, attribute, or named datatype.

Up to `size` characters of the filename are returned in `name`; additional characters, if any, are not returned to the user application.

If the length of the name, which determines the required value of `size`, is unknown, a preliminary `H5Fget_name` call can be made by setting `name` to NULL. The return value of this call will be the size of the filename; that value plus one (1) can then be assigned to `size` for a second `H5Fget_name` call, which will retrieve the actual name. (The value passed in with the parameter `size` must be one greater than size in bytes of the actual name in order to accommodate the null terminator; if `size` is set to the exact size of the name, the last byte passed back will contain the null terminator and the last character will be missing from the name passed back to the calling application.)

If an error occurs, the buffer pointed to by `name` is unchanged and the function returns a negative value.

Parameters:

hid_t `obj_id`

IN: Identifier of the object for which the associated filename is sought. The object can be a group, dataset, attribute, or named datatype.

char *`name`

OUT: Buffer to contain the returned filename.

size_t `size`

IN: Size, in bytes, of the `name` buffer.

Returns:

Returns the length of the filename if successful; otherwise returns a negative value.

Fortran90 Interface: h5fget_name_f

```
SUBROUTINE h5fget_name_f(obj_id, buf, size, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id     ! Object identifier
  CHARACTER(LEN=*), INTENT(INOUT) :: buf   ! Buffer to hold filename
  INTEGER(SIZE_T), INTENT(OUT) :: size     ! Size of the filename
  INTEGER, INTENT(OUT) :: hdferr           ! Error code: 0 on success,
                                           ! -1 if fail
END SUBROUTINE h5fget_name_f
```

History:

Release	**C**
1.6.3	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Fget_obj_count
Signature:
ssize_t `H5Fget_obj_count`(*hid_t* `file_id`, *unsigned int* `types`)
Purpose:
Returns the number of open object identifiers for an open file.
Description:
Given the identifier of an open file, `file_id`, and the desired object types, `types`, `H5Fget_obj_count` returns the number of open object identifiers for the file.

To retrieve a count of open identifiers for open objects in all HDF5 application files that are currently open, pass the value `H5F_OBJ_ALL` in `file_id`.

The types of objects to be counted are specified in `types` as follows:

`H5F_OBJ_FILE`	Files only
`H5F_OBJ_DATASET`	Datasets only
`H5F_OBJ_GROUP`	Groups only
`H5F_OBJ_DATATYPE`	Named datatypes only
`H5F_OBJ_ATTR`	Attributes only
`H5F_OBJ_ALL`	All of the above (I.e., `H5F_OBJ_FILE` | `H5F_OBJ_DATASET` | `H5F_OBJ_GROUP` | `H5F_OBJ_DATATYPE` | `H5F_OBJ_ATTR`)
`H5F_OBJ_LOCAL`	Restrict search to objects opened through current file identifier.

Multiple object types can be combined with the logical `OR` operator (`|`). For example, the expression `(H5F_OBJ_DATASET|H5F_OBJ_GROUP)` would call for datasets and groups.
Parameters:

hid_t `file_id`	IN: Identifier of a currently-open HDF5 file or `H5F_OBJ_ALL` for all currently-open HDF5 files.
unsigned int `types`	IN: Type of object for which identifiers are to be returned.

Returns:
Returns the number of open objects if successful; otherwise returns a negative value.
Fortran90 Interface: h5fget_obj_count_f
```
SUBROUTINE h5fget_obj_count_f(file_id, obj_type, obj_count, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: file_id   ! File identifier
  INTEGER, INTENT(IN)         :: obj_type  ! Object types, possible values are:
                                           !     H5F_OBJ_FILE_F
                                           !     H5F_OBJ_GROUP_F
                                           !     H5F_OBJ_DATASET_F
                                           !     H5F_OBJ_DATATYPE_F
                                           !     H5F_OBJ_ALL_F
  INTEGER, INTENT(OUT)        :: obj_count ! Number of opened objects
  INTEGER, INTENT(OUT)        :: hdferr    ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5fget_obj_count_f
```

History:

Release	**Change**
1.6.5	`H5F_OBJ_LOCAL` has been added as a qualifier on the types of objects to be counted. `H5F_OBJ_LOCAL` restricts the search to objects opened through current file identifier.
1.6.8 & 1.8.2	C function return type changed to *`ssize_t`*.

Name: H5Fget_obj_ids

Signature:

ssize_t `H5Fget_obj_ids`(*hid_t* `file_id`, *unsigned int* `types`, *size_t* `max_objs`, *hid_t* *`obj_id_list`)

Purpose:

Returns a list of open object identifiers.

Description:

Given the file identifier `file_id` and the type of objects to be identified, `types`, `H5Fget_obj_ids` returns the list of identifiers for all open HDF5 objects fitting the specified criteria.

To retrieve identifiers for open objects in all HDF5 application files that are currently open, pass the value `H5F_OBJ_ALL` in `file_id`.

The types of object identifiers to be retrieved are specified in `types` using the codes listed for the same parameter in `H5Fget_obj_count`

To retrieve identifiers for all open objects, pass a negative value for the `max_objs`.

Parameters:

hid_t `file_id`	IN: Identifier of a currently-open HDF5 file or `H5F_OBJ_ALL` for all currently-open HDF5 files.
unsigned int `types`	IN: Type of object for which identifiers are to be returned.
size_t `max_objs`	IN: Maximum number of object identifiers to place into `obj_id_list`.
hid_t *`obj_id_list`	OUT: Pointer to the returned list of open object identifiers.

Returns:

Returns number of objects placed into `obj_id_list` if successful; otherwise returns a negative value.

Fortran90 Interface: h5fget_obj_ids_f

```
SUBROUTINE h5fget_obj_ids_f(file_id, obj_type, max_objs, obj_ids, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: file_id  ! File identifier
  INTEGER,        INTENT(IN)   :: obj_type ! Object types, possible values are:
                                           !     H5F_OBJ_FILE_F
                                           !     H5F_OBJ_GROUP_F
                                           !     H5F_OBJ_DATASET_F
                                           !     H5F_OBJ_DATATYPE_F
                                           !     H5F_OBJ_ALL_F
  INTEGER, INTENT(IN)          :: max_objs ! Maximum number of object
                                           ! identifiers to retrieve
  INTEGER(HID_T), DIMENSION(*), INTENT(OUT) :: obj_ids
                                           ! Array of requested object
                                           ! identifiers
  INTEGER, INTENT(OUT)        :: hdferr    ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5fget_obj_ids_f
```

History:

Release	**Change**
1.6.0	C function introduced in this release.
1.6.8 & 1.8.2	C function return type changed to *`ssize_t`* and `max_objs` parameter datatype changed to *`size_t`*.

Name: H5Fget_vfd_handle

Signature:

herr_t `H5Fget_vfd_handle`(*hid_t* `file_id`, *hid_t* `fapl_id`, *void* **`file_handle`)

Purpose:

Returns pointer to the file handle from the virtual file driver.

Description:

Given the file identifier `file_id` and the file access property list `fapl_id`, `H5Fget_vfd_handle` returns a pointer to the file handle from the low-level file driver currently being used by the HDF5 library for file I/O.

Notes:

Users are not supposed to modify any file through this file handle.

This file handle is dynamic and is valid only while the file remains open; it will be invalid if the file is closed and reopened or opened during a subsequent session.

Parameters:

hid_t `file_id`	IN: Identifier of the file to be queried.
hid_t `fapl_id`	IN: File access property list identifier. For most drivers, the value will be `H5P_DEFAULT`. For the `FAMILY` or `MULTI` drivers, this value should be defined through the property list functions: `H5Pset_family_offset` for the `FAMILY` driver and `H5Pset_multi_type` for the `MULTI` driver.
void **`file_handle`	OUT: Pointer to the file handle being used by the low-level virtual file driver.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Fis_hdf5

Signature:

htri_t `H5Fis_hdf5`(*const char* *`name`)

Purpose:

Determines whether a file is in the HDF5 format.

Description:

`H5Fis_hdf5` determines whether a file is in the HDF5 format.

Parameters:

const char *`name`	IN: File name to check format.

Returns:

When successful, returns a positive value, for `TRUE`, or `0` (zero), for `FALSE`.
On any error, including the case that the file does not exist, returns a negative value.

Fortran90 Interface: h5fis_hdf5_f

```
SUBROUTINE h5fis_hdf5_f(name, status, hdferr)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the file
  LOGICAL, INTENT(OUT) :: status         ! This parameter indicates
                                         ! whether file is an HDF5 file
                                         ! ( TRUE or FALSE )
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5fis_hdf5_f
```

Name: H5Fmount

Signature:

herr_t `H5Fmount`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `child_id`, *hid_t* `plist_id`)

Purpose:

Mounts a file.

Description:

`H5Fmount` mounts the file specified by `child_id` onto the group specified by `loc_id` and `name` using the mount properties `plist_id`.

Note that `loc_id` is either a file or group identifier and `name` is relative to `loc_id`.

Parameters:

hid_t `loc_id`	IN: Identifier for of file or group in which `name` is defined.
const char *`name`	IN: Name of the group onto which the file specified by `child_id` is to be mounted.
hid_t `child_id`	IN: Identifier of the file to be mounted.
hid_t `plist_id`	IN: Identifier of the property list to be used.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5fmount_f

```
SUBROUTINE h5fmount_f(loc_id, name, child_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: loc_id      ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN):: name        ! Group name at locationloc_id
  INTEGER(HID_T), INTENT(IN)  :: child_id    ! File(to be mounted) identifier
  INTEGER, INTENT(OUT)        :: hdferr      ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5fmount_f
```

Last modified: 15 May 2009

Name: H5Fopen

Signature:

hid_t `H5Fopen`(*const char* *`name`, *unsigned* `flags`, *hid_t* `fapl_id`)

Purpose:

Opens an existing HDF5 file.

Description:

`H5Fopen` is the primary function for accessing existing HDF5 files. This function opens the named file in the specified access mode and with the specified access property list.

Note that `H5Fopen` does not create a file if it does not already exist; see `H5Fcreate`.

The `name` parameter specifies the name of the file to be opened.

The `fapl_id` parameter specifies the file access property list. Use of `H5P_DEFAULT` specifies that default I/O access properties are to be used

The `flags` parameter specifies whether the file will be opened in read-write or read-only mode, `H5F_ACC_RDWR` or `H5F_ACC_RDONLY`, respectively. More complex behaviors of file access are controlled through the file-access property list.

The return value is a file identifier for the open file; this file identifier should be closed by calling `H5Fclose` when it is no longer needed.

Special case -- Multiple opens:
A file can often be opened with a new `H5Fopen` call without closing an already-open identifier established in a previous `H5Fopen` or `H5Fcreate` call. Each such `H5Fopen` call will return a unique identifier and the file can be accessed through any of these identifiers as long as the identifier remains valid. In such multiply-opened cases, all the open calls should use the same `flags` argument.

In some cases, such as files on a local Unix file system, the HDF5 library can detect that a file is multiply opened and will maintain coherent access among the file identifiers.

But in many other cases, such as parallel file systems or networked file systems, it is not always possible to detect multiple opens of the same physical file. In such cases, HDF5 will treat the file identifiers as though they are accessing different files and will be unable to maintain coherent access. Errors are likely to result in these cases. While unlikely, the HDF5 library may not be able to detect, and thus report, such errors.

It is generally recommended that applications avoid multiple opens of the same file.

Parameters:

const char *`name`	IN: Name of the file to be created.

unsigned `flags`	IN: File access flags. Allowable values are:
	`H5F_ACC_RDWR`
	Allow read and write access to file.
	`H5F_ACC_RDONLY`
	Allow read-only access to file.
	◊ `H5F_ACC_RDWR` and `H5F_ACC_RDONLY` are mutually exclusive; use exactly one.

◊ An additional flag, `H5F_ACC_DEBUG`, prints debug information. This flag can be combined with one of the above values using the bit-wise OR operator (`|'), but it is used only by HDF5 Library developers; *it is neither tested nor supported* for use in applications.

hid_t `fapl_id`	IN: Identifier for the file access properties list. If parallel file access is desired, this is a collective call according to the communicator stored in the `fapl_id`. Use `H5P_DEFAULT` for default file access properties.

Returns:

Returns a file identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5fopen_f

```
SUBROUTINE h5fopen_f(name, access_flags, file_id, hdferr, &
                     access_prp)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the file
  INTEGER, INTENT(IN) :: access_flag     ! File access flags
                                         ! Possible values are:
                                         !     H5F_ACC_RDWR_F
                                         !     H5F_ACC_RDONLY_F
  INTEGER(HID_T), INTENT(OUT) :: file_id ! File identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: access_prp
                                         ! File access property list
                                         ! identifier
END SUBROUTINE h5fopen_f
```

Name: H5Freopen

Signature:

hid_t `H5Freopen`(*hid_t* `file_id`)

Purpose:

Returns a new identifier for a previously-opened HDF5 file.

Description:

`H5Freopen` returns a new file identifier for an already-open HDF5 file, as specified by `file_id`. Both identifiers share caches and other information. The only difference between the identifiers is that the new identifier is not mounted anywhere and no files are mounted on it.

Note that there is no circumstance under which `H5Freopen` can actually open a closed file; the file must already be open and have an active `file_id`. E.g., one cannot close a file with `H5Fclose (file_id)` then use `H5Freopen (file_id)` to reopen it.

The new file identifier should be closed by calling `H5Fclose` when it is no longer needed.

Parameters:

hid_t `file_id`	IN: Identifier of a file for which an additional identifier is required.

Returns:

Returns a new file identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5freopen_f

```
SUBROUTINE h5freopen_f(file_id, new_file_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: file_id     ! File identifier
  INTEGER(HID_T), INTENT(OUT) :: new_file_id ! New file identifier
  INTEGER, INTENT(OUT)        :: hdferr      ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5freopen_f
```

Name: H5Freset_mdc_hit_rate_stats

Signature:

herr_t `H5Freset_mdc_hit_rate_stats`(*hid_t* `file_id`)

Purpose:

Reset hit rate statistics counters for the target file.

Description:

H5Freset_mdc_hit_rate_stats resets the hit rate statistics counters in the metadata cache associated with the specified file.

If the adaptive cache resizing code is enabled, the hit rate statistics are reset at the beginning of each epoch. This API call allows you to do the same thing from your program.

The adaptive cache resizing code may behave oddly if you use this call when adaptive cache resizing is enabled. However, the call should be useful if you choose to control metadata cache size from your program.

See the overview of the metadata cache in the special topics section of the user manual for details of the metadata cache and the adaptive cache resizing algorithms. If you haven't read, understood, and thought about the material covered in that documentation, you shouldn't be using this API call.

Parameters:

hid_t `file_id`	IN: Identifier of the target file.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Last modified: 18 May 2009

Name: H5Fset_mdc_config

Signature:

herr_t `H5Fset_mdc_config`(*hid_t* `file_id`, *H5AC_cache_config_t* `*config_ptr`)

Purpose:

Attempt to configure metadata cache of target file.

Description:

H5Fset_mdc_config attempts to configure the file's metadata cache according configuration supplied in *config_ptr.

See the overview of the metadata cache in the special topics section of the user manual for details on what is being configured. If you haven't read and understood that documentation, you really shouldn't be using this API call.

Parameters:

hid_t `file_id`	IN: Identifier of the target file
H5AC_cache_config_t `*config_ptr`	IN: Pointer to the instance of H5AC_cache_config_t containing the desired configuration. The fields of this structure are discussed below:
General configuration section:	
int `version`	IN: Integer field indicating the the version of the H5AC_cache_config_t in use. This field should be set to H5AC__CURR_CACHE_CONFIG_VERSION (defined in H5ACpublic.h).
hbool_t `rpt_fcn_enabled`	IN: Boolean flag indicating whether the adaptive cache resize report function is enabled. This field should almost always be set to FALSE. Since resize algorithm activity is reported via stdout, it MUST be set to FALSE on Windows machines. The report function is not supported code, and can be expected to change between versions of the library. Use it at your own risk.
hbool_t `open_trace_File`	IN: Boolean field indicating whether the `trace_file_name` field should be used to open a trace file for the cache. The trace file is a debuging feature that allows the capture of top level metadata cache requests for purposes of debugging and/or optimization. This field should normally be set to `FALSE`, as trace file collection imposes considerable overhead.

	This field should only be set to `TRUE` when the `trace_file_name` contains the full path of the desired trace file, and either there is no open trace file on the cache, or the `close_trace_file` field is also `TRUE`. The trace file feature is unsupported unless used at the direction of THG. It is intended to allow THG to collect a trace of cache activity in cases of occult failures and/or poor performance seen in the field, so as to aid in reproduction in the lab. If you use it absent the direction of THG, you are on your own.
hbool_t `close_trace_file`	IN: Boolean field indicating whether the current trace file (if any) should be closed. See the above comments on the `open_trace_file` field. This field should be set to `FALSE` unless there is an open trace file on the cache that you wish to close. The trace file feature is unsupported unless used at the direction of THG. It is intended to allow THG to collect a trace of cache activity in cases of occult failures and/or poor performance seen in the field, so as to aid in reproduction in the lab. If you use it absent the direction of THG, you are on your own.
char `trace_file_name[]`	IN: Full path of the trace file to be opened if the `open_trace_file` field is `TRUE`. In the parallel case, an ascii representation of the mpi rank of the process will be appended to the file name to yield a unique trace file name for each process. The length of the path must not exceed `H5AC__MAX_TRACE_FILE_NAME_LEN` characters. The trace file feature is unsupported unless used at the direction of THG. It is intended to allow THG to collect a trace of cache activity in cases of occult failures and/or poor performance seen in the field, so as to aid in reproduction in the lab. If you use it absent the direction of THG, you are on your own.
hbool_t `evictions_enabled`	IN: A boolean flag indicating whether evictions from the metadata cache are enabled. This flag is initially set to `TRUE`.

	In rare circumstances, the raw data throughput requirements may be so high that the user wishes to postpone metadata writes so as to reserve I/O throughput for raw data. The `evictions_enabled` field exists to allow this. However, this is an extreme step, and you have no business doing it unless you have read the User Guide section on metadata caching, and have considered all other options carefully. The `evictions_enabled` field may not be set to `FALSE` unless all adaptive cache resizing code is disabled via the `incr_mode`, `flash_incr_mode`, and `decr_mode` fields. When this flag is set to `FALSE`, the metadata cache will not attempt to evict entries to make space for new entries, and thus will grow without bound. Evictions will be re-enabled when this field is set back to `TRUE`. This should be done as soon as possible.
hbool_t `set_initial_size`	IN: Boolean flag indicating whether the cache should be forced to the user specified initial size.
size_t `initial_size`	IN: If set_initial_size is TRUE, initial_size must contains the desired initial size in bytes. This value must lie in the closed interval [min_size, max_size]. (see below)
double `min_clean_fraction`	IN: This field specifies the minimum fraction of the cache that must be kept either clean or empty. The value must lie in the interval [0.0, 1.0]. 0.01 is a good place to start in the serial case. In the parallel case, a larger value is needed -- see the overview of the metadata cache in the â’ HDF5 Special Topicsâ’ section of the *HDF5 Userâ’ s Guide* for details.
size_t `max_size`	IN: Upper bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
size_t `min_size`	IN: Lower bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
long int `epoch_length`	IN: Number of cache accesses between runs of the adaptive cache resize code. 50,000 is a good starting number.

Increment configuration section:

enum H5C_cache_incr_mode `incr_mode`	IN: Enumerated value indicating the operational mode of the automatic cache size increase code. At present, only two values are legal: H5C_incr__off: Automatic cache size increase is disabled, and the remaining increment fields are ignored. H5C_incr__threshold: Automatic cache size increase is enabled using the hit rate threshold algorithm.
double `lower_hr_threshold`	IN: Hit rate threshold used by the hit rate threshold cache size increment algorithm. When the hit rate over an epoch is below this threshold and the cache is full, the maximum size of the cache is multiplied by increment (below), and then clipped as necessary to stay within max_size, and possibly max_increment. This field must lie in the interval [0.0, 1.0]. 0.8 or 0.9 is a good starting point.
double `increment`	IN: Factor by which the hit rate threshold cache size increment algorithm multiplies the current cache max size to obtain a tentative new cache size. The actual cache size increase will be clipped to satisfy the max_size specified in the general configuration, and possibly max_increment below. The parameter must be greater than or equal to 1.0 -- 2.0 is a reasonable value. If you set it to 1.0, you will effectively disable cache size increases.
hbool_t `apply_max_increment`	IN: Boolean flag indicating whether an upper limit should be applied to the size of cache size increases.
size_t `max_increment`	IN: Maximum number of bytes by which cache size can be increased in a single step -- if applicable.
enum H5C_cache_flash_incr_mode `flash_incr_mode`	IN: Enumerated value indicating the operational mode of the flash cache size increase code. At present, only the following values are legal:

	H5C_flash_incr__off: Flash cache size increase is disabled. H5C_flash_incr__add_space: Flash cache size increase is enabled using the add space algorithm.
double `flash_threshold`	IN: The factor by which the current maximum cache size is multiplied to obtain the minimum size entry / entry size increase which may trigger a flash cache size increase. At present, this value must lie in the range [0.1, 1.0].
double `flash_multiple`	IN: The factor by which the size of the triggering entry / entry size increase is multiplied to obtain the initial cache size increment. This increment may be reduced to reflect existing free space in the cache and the `max_size` field above. At present, this field must lie in the range [0.1, 10.0].
Decrement configuration section:	
enum H5C_cache_decr_mode `decr_mode`	IN: Enumerated value indicating the operational mode of the automatic cache size decrease code. At present, the following values are legal: H5C_decr__off: Automatic cache size decrease is disabled. H5C_decr__threshold: Automatic cache size decrease is enabled using the hit rate threshold algorithm. H5C_decr__age_out: Automatic cache size decrease is enabled using the ageout algorithm. H5C_decr__age_out_with_threshold: Automatic cache size decrease is enabled using the ageout with hit rate threshold algorithm

double `upper_hr_threshold`	IN: Hit rate threshold for the hit rate threshold and ageout with hit rate threshold cache size decrement algorithms. When decr_mode is H5C_decr__threshold, and the hit rate over a given epoch exceeds the supplied threshold, the current maximum cache size is multiplied by decrement to obtain a tentative new (and smaller) maximum cache size. When decr_mode is H5C_decr__age_out_with_threshold, there is no attempt to find and evict aged out entries unless the hit rate in the previous epoch exceeded the supplied threshold. This field must lie in the interval [0.0, 1.0]. For H5C_incr__threshold, .9995 or .99995 is a good place to start. For H5C_decr__age_out_with_threshold, .999 might be more useful.
double `decrement`	IN: In the hit rate threshold cache size decrease algorithm, this parameter contains the factor by which the current max cache size is multiplied to produce a tentative new cache size. The actual cache size decrease will be clipped to satisfy the min_size specified in the general configuration, and possibly max_decrement below. The parameter must be be in the interval [0.0, 1.0]. If you set it to 1.0, you will effectively disable cache size decreases. 0.9 is a reasonable starting point.
hbool_t `apply_max_decrement`	IN: Boolean flag indicating whether an upper limit should be applied to the size of cache size decreases.
size_t `max_decrement`	IN: Maximum number of bytes by which the maximum cache size can be decreased in any single step -- if applicable.

int `epochs_before_eviction`	IN: In the ageout based cache size reduction algorithms, this field contains the minimum number of epochs an entry must remain unaccessed in cache before the cache size reduction algorithm tries to evict it. 3 is a reasonable value.
hbool_t `apply_empty_reserve`	IN: Boolean flag indicating whether the ageout based decrement algorithms will maintain a empty reserve when decreasing cache size.
double `empty_reserve`	IN: Empty reserve as a fraction of maximum cache size if applicable. When so directed, the ageout based algorithms will not decrease the maximum cache size unless the empty reserve can be met. The parameter must lie in the interval [0.0, 1.0]. 0.1 or 0.05 is a good place to start.
Parallel configuration section:	
int `dirty_bytes_threshold`	IN: Threshold number of bytes of dirty metadata generation for triggering synchronizations of the metadata caches serving the target file in the parallel case. Synchronization occurs whenever the number of bytes of dirty metadata created since the last synchronization exceeds this limit. This field only applies to the parallel case. While it is ignored elsewhere, it can still draw a value out of bounds error. It must be consistant across all caches on any given file. By default, this field is set to 256 KB. It shouldn't be more than half the current max cache size times the min clean fraction.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Name: H5Funmount
Signature:
herr_t `H5Funmount`(*hid_t* `loc_id`, *const char* *`name`)
Purpose:
Unmounts a file.
Description:
Given a mount point, `H5Funmount` dissassociates the mount point's file from the file mounted there. This function does not close either file.

The mount point can be either the group in the parent or the root group of the mounted file (both groups have the same name). If the mount point was opened before the mount then it is the group in the parent; if it was opened after the mount then it is the root group of the child.

Note that `loc_id` is either a file or group identifier and `name` is relative to `loc_id`.
Parameters:

hid_t `loc_id`	IN: File or group identifier for the location at which the specified file is to be unmounted.
const char *`name`	IN: Name of the mount point.

Returns:
Returns a non-negative value if successful; otherwise returns a negative value.
Fortran90 Interface: h5funmount_f
```
SUBROUTINE h5funmount_f(loc_id, name, child_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: loc_id      ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN):: name        ! Group name at location loc_id
  INTEGER, INTENT(OUT)        :: hdferr      ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5funmount_f
```

Last modified: 25 April 2009

H5G: Group Interface

Group Object API Functions

The Group interface functions create and manipulate groups of objects in an HDF5 file.
In the following lists, italic type indicates a configurable macro.

The C Interfaces:

• *H5Gcreate*	• H5Gmove *	• H5Gget_objinfo *
• H5Gcreate1 *	• H5Gmove2 *	• H5Gget_num_objs *
• H5Gcreate2	• H5Glink *	• H5Gget_create_plist
• H5Gcreate_anon	• H5Glink2 *	• H5Gget_linkval *
• *H5Gopen*	• H5Gunlink *	• H5Giterate *
• H5Gopen1 *	• H5Gset_comment *	• H5Gget_info_by_idx
• H5Gopen2	• H5Gget_comment *	• H5Gget_objname_by_idx *
• H5Gclose	• H5Gget_info	• H5Gget_objtype_by_idx *
	• H5Gget_info_by_name	

** Use of these functions is deprecated in Release 1.8.0.*

Alphabetical Listing

• H5Gclose	• H5Gget_info	• H5Glink2 *
• *H5Gcreate*	• H5Gget_info_by_idx	• H5Gmove *
• H5Gcreate1 *	• H5Gget_info_by_name	• H5Gmove2 *
• H5Gcreate2	• H5Gget_num_objs *	• *H5Gopen*
• H5Gcreate_anon	• H5Gget_objinfo *	• H5Gopen1 *
• H5Gget_comment *	• H5Gget_objname_by_idx *	• H5Gopen2
• H5Gget_create_plist	• H5Gget_objtype_by_idx *	• H5Gset_comment *
• H5Gget_linkval *	• H5Giterate *	• H5Gunlink *
	• H5Glink *	

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

• h5gclose_f	• h5gget_info_f	• h5glink2_f *
• *h5gcreate_f*	• h5gget_info_by_idx_f	• h5gmove_f *
• h5gcreate_anon_f	• h5gget_info_by_name_f	• h5gmove2_f *
• h5gget_comment_f *	• h5giterate_f *	• *h5gopen_f*
• h5gget_create_plist_f	• h5glink_f *	• h5gset_comment_f *
• h5gget_linkval_f *		• h5gunlink_f *

** Use of these functions is deprecated in Release 1.8.0.*

Groups in HDF5:
A group associates names with objects and provides a mechanism for mapping a name to an object. Since all objects appear in at least one group (with the possible exception of the root object) and since objects can have names in more than one group, the set of all objects in an HDF5 file is a directed graph. The internal nodes (nodes with out-degree greater than zero) must be groups while the leaf nodes (nodes with out-degree zero) are either empty groups or objects of some other type. Exactly one object in every non-empty file is the root object. The root object always has a positive in-degree because it is pointed to by the file super block.

Group implementations in HDF5:
The original HDF5 group implementation provided a single indexed structure for link storage. A new group implementation, in HDF5 Release 1.8.0, enables more efficient compact storage for very small groups, improved link indexing for large groups, and other advanced features.

- The *original indexed* format remains the default. Links are stored in a B-tree in the group's local heap.
- Groups created in the new *compact-or-indexed* format, the implementation introduced with Release 1.8.0, can be tuned for performance, switching between the compact and indexed formats at thresholds set in the user application.
    - The *compact* format will conserve file space and processing overhead when working with small groups and is particularly valuable when a group contains no links. Links are stored as a list of messages in the group's header.
    - The *indexed* format will yield improved performance when working with large groups, e.g., groups containing thousands to millions of members. Links are stored in a fractal heap and indexed with an improved B-tree.
- The new implementation also enables the use of link names consisting of non-ASCII character sets (see `H5Pset_char_encoding`) and is required for all link types other than hard or soft links, e.g., external and user-defined links (see the H5L APIs).

The original group structure and the newer structures are not directly interoperable. By default, a group will be created in the original indexed format. An existing group can be changed to a compact-or-indexed format if the need arises; there is no capability to change back. As stated above, once in the compact-or-indexed format, a group can switch between compact and indexed as needed.

Groups will be initially created in the compact-or-indexed format only when one or more of the following conditions is met:

- The *low version bound* value of the *library version bounds* property has been set to Release 1.8.0 or later in the file access property list (see `H5Pset_libver_bounds`). Currently, that would require an `H5Pset_libver_bounds` call with the *low* parameter set to `H5F_LIBVER_LATEST`.

  When this property is set for an HDF5 file, all objects in the file will be created using the latest available format; no effort will be made to create a file that can be read by older libraries.
- The creation order tracking property, `H5P_CRT_ORDER_TRACKED`, has been set in the group creation property list (see `H5Pset_link_creation_order`).

An existing group, currently in the original indexed format, will be converted to the compact-or-indexed format upon the occurrence of any of the following events:

- An external or user-defined link is inserted into the group.
- A link named with a string composed of non-ASCII characters is inserted into the group.

The compact-or-indexed format offers performance improvements that will be most notable at the extremes, i.e., in groups with zero members and in groups with tens of thousands of members. But measurable differences may sometimes appear at a threshold as low as eight group members. Since these performance thresholds and criteria differ from application to application, tunable settings are provided to govern the switch between the compact and indexed formats (see `H5Pset_link_phase_change`). Optimal thresholds will depend on the application and the operating environment.

Future versions of HDF5 will retain the ability to create, read, write, and manipulate all groups stored in either the original indexed format or the compact-or-indexed format.

Locating objects in the HDF5 file hierarchy:
An object name consists of one or more components separated from one another by slashes. An absolute name begins with a slash and the object is located by looking for the first component in the root object, then looking for the second component in the first object, etc., until the entire name is traversed. A relative name does not begin with a slash and the traversal begins at the location specified by the create or access function.

Name: H5Gclose

Signature:

herr_t `H5Gclose`(*hid_t* `group_id`)

Purpose:

Closes the specified group.

Description:

`H5Gclose` releases resources used by a group which was opened by `H5Gcreate*` or `H5Gopen*`. After closing a group, the `group_id` cannot be used again.

Failure to release a group with this call will result in resource leaks.

Parameters:

hid_t `group_id`	IN: Group identifier to release.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gclose_f

```
SUBROUTINE h5gclose_f( gr_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gr_id     ! Group identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5gclose_f
```

Name: H5Gcreate
Signatures:

hid_t `H5Gcreate`(*hid_t* `loc_id`, *const char* `*name`, *size_t* `size_hint`) [1]

hid_t `H5Gcreate`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `lcpl_id`, *hid_t* `gcpl_id`, *hid_t* `gapl_id`) [2]

Purpose:

Creates a new empty group and links it to a location in the file.

Description:

`H5Gcreate` is a macro that is mapped to either `H5Gcreate1` or `H5Gcreate2`, depending on the HDF5 Library configuration and application compile-time Compatibility Macro Mapping Options.

This macro is provided to facilitate application compatibility. For example:

- ◊ The `H5Gcreate` macro will be mapped to `H5Gcreate1` and will use the `H5Gcreate1` syntax (first signature above) if the application is coded for HDF5 Release 1.6.*x*.
- ◊ The `H5Gcreate` macro will be mapped to `H5Gcreate2` and will use the `H5Gcreate2` syntax (second signature above) if the application is coded for HDF5 Release 1.8.*x*.

Macro use and Compatibility Macro Mapping Options are fully described in “API Compatibility Macros in HDF5”.

When both the HDF5 Library and the application are built without specific Compatibility Macro Mapping Options, the default behavior occurs and `H5Gcreate` is mapped to the most recent version of the function, currently `H5Gcreate2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Gcreate` will be mapped to `H5Gcreate1`.

Function Mapping flags can be used to override these settings on a function-by-function basis when the application is compiled. The `H5Gcreate` Function Mapping flags are shown:

`h5cc` flag	macro maps to
`-D H5Acreate_vers=1`	`H5Acreate1`
`-D H5Acreate_vers=2`	`H5Acreate2`

Interface history: Signature [1] above is the original `H5Gcreate` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Gcreate1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Gcreate2`.

Deprecated functions may not be available in all installations of the HDF5 libary. See “API Compatibility Macros in HDF5” for details.

Fortran90 Interface: h5gcreate_f

```
SUBROUTINE h5gcreate_f(loc_id, name, grp_id, hdferr, &
                       size_hint, lcpl_id, gcpl_id, gapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the group
  INTEGER(HID_T), INTENT(OUT) :: grp_id  ! Group identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
  INTEGER(SIZE_T), OPTIONAL, INTENT(IN) :: size_hint
                                         ! Parameter indicating the number of
                                         ! bytes to reserve for the names that
                                         ! will appear in the group.
                                         ! Note, set to OBJECT_NAMELEN_DEFAULT_F
                                         ! if using any of the optional
                                         ! parameters lcpl_id, gcpl_id,
                                         ! and/or gapl_id when not
                                         ! using keywords in specifying the
                                         ! optional parameters.
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                                         ! Property list for link creation
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: gcpl_id
                                         ! Property list for group creation
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: gapl_id
                                         ! Property list for group access
END SUBROUTINE h5gcreate_f
```

History:

Release	**C**
1.8.0	The function `H5Gcreate` renamed to `H5Gcreate1` and deprecated in this release. The macro `H5Gcreate` and the function `H5Gcreate2` introduced in this release.

Last modified: 18 May 2009

Name: H5Gcreate1

Signature:

hid_t `H5Gcreate1`(*hid_t* `loc_id`, *const char* *`name`, *size_t* `size_hint`)

Purpose:

Creates a new empty group and links it to a location in the file.

Notice:

This function is renamed from `H5Gcreate` and deprecated in favor of the functions `H5Gcreate2` and `H5Gcreate_anon`, or the new macro `H5Gcreate`.

Description:

`H5Gcreate1` creates a new group with the specified name at the specified location, `loc_id`. The location is identified by a file or group identifier. The name, `name`, must not already be taken by some other object and all parent groups must already exist.

`name` can be a relative path based at `loc_id` or an absolute path from the root of the file. If any of the groups specified in that path do not already exist, the group must be created with `H5Gcreate_anon` and linked into the file structure with `H5Lcreate_hard`.

The length of a group name, or of the name of any object within a group, is not limited.

`size_hint` is a hint for the number of bytes to reserve to store the names which will be eventually added to the new group. Passing a value of zero for `size_hint` is usually adequate since the library is able to dynamically resize the name heap, but a correct hint may result in better performance. If a non-positive value is supplied for `size_hint`, then a default size is chosen.

The return value is a group identifier for the open group. This group identifier should be closed by calling `H5Gclose` when it is no longer needed.

See `H5Gcreate_anon` for a discussion of the differences between `H5Gcreate1` and `H5Gcreate_anon`.

Parameters:

hid_t `loc_id`	IN: File or group identifier.
const char *`name`	IN: Absolute or relative name of the o new group.
size_t `size_hint`	IN: Optional parameter indicating the number of bytes to reserve for the names that will appear in the group. A conservative estimate could result in multiple system-level I/O requests to read the group name heap; a liberal estimate could result in a single large I/O request even when the group has just a few names. HDF5 stores each name with a null terminator.

Returns:

Returns a valid group identifier for the open group if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Gcreate`.

History:

Release	**C**
1.8.0	Function `H5Gcreate` renamed to `H5Gcreate1` and deprecated in this release.

Name: H5Gcreate2

Signature:

hid_t `H5Gcreate2`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `lcpl_id`, *hid_t* `gcpl_id`, *hid_t* `gapl_id`)

Purpose:

Creates a new empty group and links it into the file.

Description:

`H5Gcreate2` creates a new group named `name` at the location specified by `loc_id` with the group creation and access properties spceified in `gcpl_id` and `gapl_id`, respectively.

`loc_id` may be a file identifier, or a group identifier within that file. `name` may be either an absolute path in the file or a relative path from `loc_id` naming the dataset.

The link creation property list, `lcpl_id`, governs creation of the link(s) by which the new dataset is accessed and the creation of any intermediate groups that may be missing.

To conserve and release resources, the group should be closed when access is no longer required.

Parameters:

hid_t `loc_id`	IN: File or group identifier
const char `*name`	IN: Absolute or relative name of the new group
hid_t `lcpl_id`	IN: Property list for link creation
hid_t `gcpl_id`	IN: Property list for group creation
hid_t `gapl_id`	IN: Property list for group access (No group access properties have been implemented at this time; use `H5P_DEFAULT`.)

Returns:

Returns a group identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Gcreate`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Last modified: 18 May 2009

Name: H5Gcreate_anon

Signature:

hid_t `H5Gcreate_anon`(*hid_t* `loc_id`, *hid_t* `gcpl_id`, *hid_t* `gapl_id`)

Purpose:

Creates a new empty group without linking it into the file structure.

Description:

`H5Gcreate_anon` creates a new empty group in the file specified by `loc_id`. With default settings, `H5Gcreate_anon` provides similar functionality to that provided by `H5Gcreate`, with the differences described below.

The new group's creation and access properties are specified in `gcpl_id` and `gapl_id`, respectively.

`H5Gcreate_anon` returns a new group identifier. This identifier *must* be linked into the HDF5 file structure with `H5Lcreate_hard` or it will be deleted from the file when the file is closed.

The differences between this function and `H5Gcreate1` are as follows:

◊ `H5Gcreate1` does not provide for the use of custom property lists; `H5Gcreate1` always uses default properties.

◊ `H5Gcreate_anon` neither provides the new group's name nor links it into the HDF5 file structure; those actions must be performed separately through a call to `H5Lcreate_hard`, which offers greater control over linking.

◊ `H5Gcreate_anon` does not directly provide a *hint* mechanism for the group's heap size. Comparable information can be included in the group creation property list `gcpl_id` through a `H5Pset_local_heap_size_hint` call.

Parameters:

hid_t `loc_id`	File or group identifier specifying the file in which the new group is to be created
hid_t `gcpl_id`	Group creation property list identifier (`H5P_DEFAULT` for the default property list)
hid_t `gapl_id`	Group access property list identifier (No group access properties have been implemented at this time; use `H5P_DEFAULT`.)

Returns:

Returns a new group identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5gcreate_anon_f

```
SUBROUTINE h5gcreate_anon_f(loc_id, grp_id, hdferr, gcpl_id, gapl_id)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! File or group identifier
  INTEGER(HID_T), INTENT(OUT) :: grp_id  ! Group identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: gcpl_id
                                         ! Property list for group creation
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: gapl_id
                                         ! Property list for group access
END SUBROUTINE h5gcreate_anon_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Gget_comment

Signature:

int `H5Gget_comment`(*hid_t* `loc_id`, *const char* *`name`, *size_t* `bufsize`, *char* *`comment`)

Purpose:

Retrieves comment for specified object.

Notice:

This function is deprecated in favor of the function `H5Oget_comment`.

Description:

`H5Gget_comment` retrieves the comment for the the object specified by `loc_id` and `name`. The comment is returned in the buffer `comment`.

`loc_id` can specify any object in the file. `name` can be one of the following:
— The name of the object relative to `loc_id`
— An absolute name of the object, starting from `/`, the file's root group
— A dot (`.`), if `loc_id` fully specifies the object

At most `bufsize` characters, including a null terminator, are returned in `comment`. The returned value is not null terminated if the comment is longer than the supplied buffer. If the size of the comment is unknown, a preliminary `H5Gget_comment` call will return the size of the comment, including space for the null terminator.

If an object does not have a comment, the empty string is returned in `comment`.

Parameters:

hid_t `loc_id`	IN: Identifier of the file, group, dataset, or named datatype.
const char *`name`	IN: Name of the object in `loc_id` whose comment is to be retrieved. `name` must be '.' (dot) if `loc_id` fully specifies the object for which the associated comment is to be retrieved.
size_t `bufsize`	IN: Anticipated required size of the `comment` buffer.
char *`comment`	OUT: The comment.

Returns:

Returns the number of characters in the comment, counting the null terminator, if successful; the value returned may be larger than `bufsize`. Otherwise returns a negative value.

Fortran90 Interface: h5gget_comment_f

```
SUBROUTINE h5gget_comment_f(loc_id, name, size, buffer, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id         ! File, group, dataset, or
                                               ! named datatype identifier
  CHARACTER(LEN=*), INTENT(IN) :: name         ! Name of the object link
  CHARACTER(LEN=size), INTENT(OUT) :: buffer   ! Buffer to hold the comment
  INTEGER, INTENT(OUT) :: hdferr               ! Error code
                                               ! 0 on success and -1 on failure
END SUBROUTINE h5gget_comment_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Gget_create_plist

Signature:

hid_t `H5Gget_create_plist`(*hid_t* `group_id`)

Purpose:

Gets a group creation property list identifier.

Description:

`H5Gget_create_plist` returns an identifier for the group creation property list associated with the group specified by `group_id`.

The creation property list identifier should be released with `H5Pclose`.

Parameters:

hid_t `group_id`	IN: Identifier of the group.

Returns:

Returns an identifier for the group's creation property list if successful. Otherwise returns a negative value.

Fortran90 Interface: h5gget_create_plist_f

```
SUBROUTINE h5gget_create_plist_f(grp_id, gcpl_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: grp_id  ! Group identifier
  INTEGER(HID_T), INTENT(OUT) :: gcpl_id ! Property list for group creation
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5gget_create_plist_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Gget_info

Signature:

herr_t `H5Gget_info`(*hid_t* `group_id`, *H5G_info_t* *`group_info`)

Purpose:

Retrieves information about a group.

Description:

`H5Gget_info` retrieves information about the group specified by `group_id`. The information is returned in the `group_info` struct.

`group_info` is an *H5G_info_t* struct and is defined (in `H5Gpublic.h`) as follows:

H5G_storage_type_t `storage_type`	Type of storage for links in group `H5G_STORAGE_TYPE_COMPACT`: Compact storage `H5G_STORAGE_TYPE_DENSE`: Indexed storage `H5G_STORAGE_TYPE_SYMBOL_TABLE`: Symbol tables, the original HDF5 structure
hsize_t `nlinks`	Number of links in group
int64_t `max_corder`	Current maximum creation order value for group
hbool_t `mounted`	Whether the group has a file mounted on it

Parameters:

hid_t `group_id`	IN: Group identifier
H5G_info_t *`group_info`	OUT: Struct in which group information is returned

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gget_info_f

```
SUBROUTINE h5gget_info_f(group_id, storage_type, nlinks, max_corder, hdferr, &
                         mounted)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: group_id
                             ! Group identifier
  INTEGER, INTENT(OUT) :: storage_type
                             ! Type of storage for links in group:
                             !    H5G_STORAGE_TYPE_COMPACT_F: Compact storage
                             !    H5G_STORAGE_TYPE_DENSE_F: Indexed storage
                             !    H5G_STORAGE_TYPE_SYMBOL_TABLE_F: Symbol tables
  INTEGER, INTENT(OUT) :: nlinks
                             ! Number of links in group
  INTEGER, INTENT(OUT) :: max_corder
                             ! Current maximum creation order value for group
  INTEGER, INTENT(OUT) :: hdferr
                             ! Error code:
                             ! 0 on success and -1 on failure
  LOGICAL, INTENT(OUT), OPTIONAL :: mounted
                             ! Whether group has a file mounted on it
END SUBROUTINE h5gget_info_f
```

History:

Release	**C**
1.8.2	Added 'mounted' field.
1.8.0	Function introduced in this release.

Name: H5Gget_info_by_idx

Signature:

herr_t `H5Gget_info_by_idx`(*hid_t* `loc_id`, *const char* `*group_name`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *hsize_t* `n`, *H5G_info_t* `*group_info`, *hid_t* `lapl_id`)

Purpose:

Retrieves information about a group, according to the group's position within an index.

Description:

`H5Gget_info_by_idx` retrieves the same imformation about a group as retrieved by the function `H5Gget_info`, immediately above, but the means of identifying the group differs; the group is identified by position in an index rather than by name.

`loc_id` and `group_name` specify the group containing the group for which information is sought. The groups in `group_name` are indexed by `index_type`; the group for which information is retrieved is identified in that index by index order, `order`, and index position, `n`.

If `loc_id` specifies the group containing the group for which information is queried, `group_name` can be a dot (.).

Valid values for `index_type` are as follows:

`H5_INDEX_NAME`	An alpha-numeric index by group name
`H5_INDEX_CRT_ORDER`	An index by creation order

The order in which the index is to be examined, as specified by `order`, can be one of the following:

`H5_ITER_INC`	The count is from beginning of the index, i.e., top-down.
`H5_ITER_DEC`	The count is from the end of the index, i.e., bottom-up.
`H5_ITER_NATIVE`	HDF5 counts through the index in the fastest-available order. No information is provided as to the order, but HDF5 ensures that no element in the index will be overlooked.

Parameters:

hid_t `loc_id`	IN: File or group identifier
const char `*group_name`	IN: Name of group containing group for which information is to be retrieved
H5_index_t `index_type`	IN: Index type
H5_iter_order_t `order`	IN: Order of the count in the index
hsize_t `n`	IN: Position in the index of the group for which information is retrieved
H5G_info_t `*group_info`	OUT: Struct in which group information is returned
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gget_info_by_idx_f

```
SUBROUTINE h5gget_info_by_idx_f(loc_id, group_name, index_type, order, n, &
     storage_type, nlinks, max_corder, hdferr, lapl_id, mounted)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                             ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: group_name
                             ! Name of group containing group for which
                             ! information is to be retrieved
  INTEGER, INTENT(IN) :: index_type
                             ! Index type
  INTEGER, INTENT(IN) :: order
                             ! Order of the count in the index
  INTEGER(HSIZE_T), INTENT(IN) :: n
                             ! Position in the index of the group for which
                             ! information is retrieved
  INTEGER, INTENT(OUT) :: storage_type
                             ! Type of storage for links in group:
                             ! H5G_STORAGE_TYPE_COMPACT_F: Compact storage
                             ! H5G_STORAGE_TYPE_DENSE_F: Indexed storage
                             ! H5G_STORAGE_TYPE_SYMBOL_TABLE_F: Symbol tables
  INTEGER, INTENT(OUT) :: nlinks
                             ! Number of links in group
  INTEGER, INTENT(OUT) :: max_corder
                             ! Current maximum creation order value for group
  INTEGER, INTENT(OUT) :: hdferr
                             ! Error code:
                             ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                             ! Link access property list
  LOGICAL, INTENT(OUT), OPTIONAL :: mounted
                             ! Whether group has a file mounted on it
END SUBROUTINE h5gget_info_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Gget_info_by_name

Signature:

herr_t `H5Gget_info_by_name`(*hid_t* `loc_id`, *const char* *`group_name`, *H5G_info_t* *`group_info`, *hid_t* `lapl_id`)

Purpose:

Retrieves information about a group.

Description:

`H5Gget_info_by_name` retrieves information about the group `group_name` located in the file or group specified by `loc_id`. The information is returned in the `group_info` struct.

If `loc_id` specifies the group for which information is queried, `group_name` can be a dot (.).

`group_info` is an *H5G_info_t* struct and is defined (in `H5Gpublic.h`) as follows:

H5G_storage_type_t `storage_type`	Type of storage for links in group `H5G_STORAGE_TYPE_COMPACT`: Compact storage `H5G_STORAGE_TYPE_DENSE`: Dense storage `H5G_STORAGE_TYPE_SYMBOL_TABLE`: Symbol tables, the original HDF5 structure
hsize_t `nlinks`	Number of links in group
int64_t `max_corder`	Current maximum creation order value for group
hbool_t `mounted`	Whether the group has a file mounted on it

Parameters:

hid_t `loc_id`	IN: File or group identifier
const char *`group_name`	IN: Name of group for which information is to be retrieved
H5G_info_t *`group_info`	OUT: Struct in which group information is returned
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gget_info_by_name_f

```
SUBROUTINE h5gget_info_by_name_f(loc_id, group_name, &
     storage_type, nlinks, max_corder, hdferr, lapl_id, mounted)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                               ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: group_name
                               ! Name of group containing group for which
                               ! information is to be retrieved
  INTEGER, INTENT(OUT) :: storage_type
                               ! Type of storage for links in group:
                               !   H5G_STORAGE_TYPE_COMPACT_F: Compact storage
                               !   H5G_STORAGE_TYPE_DENSE_F: Indexed storage
                               !   H5G_STORAGE_TYPE_SYMBOL_TABLE_F: Symbol tables
  INTEGER, INTENT(OUT) :: nlinks
                               ! Number of links in group
  INTEGER, INTENT(OUT) :: max_corder
                               ! Current maximum creation order value for group
```

```
    INTEGER, INTENT(OUT) :: hdferr
                               ! Error code:
                               ! 0 on success and -1 on failure
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                               ! Link access property list
    LOGICAL, INTENT(OUT), OPTIONAL :: mounted
                                 ! Whether group has a file mounted on it
  END SUBROUTINE h5gget_info_by_name_f
```

History:

Release	**C**
1.8.2	Added 'mounted' field.
1.8.0	Function introduced in this release.

Name: H5Gget_linkval

Signature:

herr_t `H5Gget_linkval`(*hid_t* `loc_id`, *const char* *`name`, *size_t* `size`, *char* *`value`)

Purpose:

Returns the name of the object that the symbolic link points to.

Notice:

This function is deprecated in favor of the function `H5Lget_val`.

Description:

`H5Gget_linkval` returns `size` characters of the name of the object that the symbolic link `name` points to.

The parameter `loc_id` is a file or group identifier.

The parameter `name` must be a symbolic link pointing to the desired object and must be defined relative to `loc_id`.

If `size` is smaller than the size of the returned object name, then the name stored in the buffer `value` will not be null terminated.

This function fails if `name` is not a symbolic link. The presence of a symbolic link can be tested by passing zero for `size` and NULL for `value`.

This function should be used only after `H5Lget_info` (or the deprecated function `H5Gget_objinfo`) has been called to verify that `name` is a symbolic link.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group.
const char *`name`	IN: Symbolic link to the object whose name is to be returned.
size_t `size`	IN: Maximum number of characters of `value` to be returned.
char *`value`	OUT: A buffer to hold the name of the object being sought.

Returns:

Returns a non-negative value, with the link value in `value`, if successful. Otherwise returns a negative value.

Fortran90 Interface: h5gget_linkval_f

```
SUBROUTINE h5gget_linkval_f(loc_id, name, size, buffer, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id         ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name         ! Name of the symbolic link
  CHARACTER(LEN=size), INTENT(OUT) :: buffer   ! Buffer to hold a
                                               ! name of the object
                                               ! symbolic link points to
  INTEGER, INTENT(OUT) :: hdferr               ! Error code
                                               ! 0 on success and -1 on failure
END SUBROUTINE h5gget_linkval_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Gget_num_objs

Signature:

herr_t `H5Gget_num_objs`(*hid_t* `loc_id`, *hsize_t** `num_obj`)

Purpose:

Returns number of objects in the group specified by its identifier

Notice:

This function is deprecated in favor of the function *`H5Gget_info`*.

Description:

`H5Gget_num_objs` returns number of objects in a group. Group is specified by its identifier `loc_id`. If a file identifier is passed in, then the number of objects in the root group is returned.

Parameters:

hid_t `loc_id`	IN: Identifier of the group or the file
hsize_t *`num_obj`	OUT: Number of objects in the group.

Returns:

Returns positive value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.
1.8.0	Function deprecated in this release.

Name: H5Gget_objinfo

Signature:

herr_t `H5Gget_objinfo`(*hid_t* `loc_id`, *const char* `*name`, *hbool_t* `follow_link`, *H5G_stat_t* `*statbuf`)

Purpose:

Returns information about an object.

Notice:

This function is deprecated in favor of the function `H5Oget_info` and `H5Lget_info`.

Description:

`H5Gget_objinfo` returns information about the specified object through the `statbuf` argument.

A file or group identifier, `loc_id`, and an object name, `name`, relative to `loc_id`, are commonly used to specify the object. However, if the object identifier is already known to the application, an alternative approach is to use that identifier, `obj_id`, in place of `loc_id`, and a dot (`.`) in place of `name`. Thus, the alternative versions of the first portion of an `H5Gget_objinfo` call would be as follows:

```
     H5Gget_objinfo (loc_id name  ...)
    H5Gget_objinfo (obj_id .     ...)
```

If the object is a symbolic link and `follow_link` is zero (`0`), then the information returned describes the link itself; otherwise the link is followed and the information returned describes the object to which the link points. If `follow_link` is non-zero but the final symbolic link is dangling (does not point to anything), then an error is returned. The `statbuf` fields are undefined for an error. The existence of an object can be tested by calling this function with a null `statbuf`.

`H5Gget_objinfo` fills in the following data structure (defined in H5Gpublic.h):

```
typedef struct H5G_stat_t {
    unsigned long fileno[2];
    haddr_t objno[2];
    unsigned nlink;
    H5G_obj_t type;
    time_t mtime;
    size_t linklen;
    H5O_stat_t ohdr;
} H5G_stat_t
```

where H5O_stat_t (defined in H5Opublic.h) is:

```
typedef struct H5O_stat_t {
    hsize_t size;
    hsize_t free;
    unsigned nmesgs;
    unsigned nchunks;
} H5O_stat_t
```

The `fileno` and `objno` fields contain four values which uniquely identify an object among those HDF5 files which are open: if all four values are the same between two objects, then the two objects are the same (provided both files are still open).

◊ Note that if a file is closed and re-opened, the value in `fileno` will change.

◊ If a VFL driver either does not or cannot detect that two `H5Fopen` calls referencing the same file actually open the same file, each will get a different `fileno`.

The `nlink` field is the number of hard links to the object or zero when information is being returned about a symbolic link (symbolic links do not have hard links but all other objects always have at least one).

The `type` field contains the type of the object, one of `H5G_GROUP`, `H5G_DATASET`, `H5G_LINK`, or `H5G_TYPE`.

The `mtime` field contains the modification time.

If information is being returned about a symbolic link then `linklen` will be the length of the link value (the name of the pointed-to object with the null terminator); otherwise `linklen` will be zero.

The fields in the `H5O_stat_t` struct contain information about the object header for the object queried:

`size`	The total size of all the object header information in the file (for all chunks).
`free`	The size of unused space in the object header.
`nmesgs`	The number of object header messages.
`nchunks`	The number of chunks the object header is broken up into.

Other fields may be added to this structure in the future.

Note:

Some systems will be able to record the time accurately but unable to retrieve the correct time; such systems (e.g., Irix64) will report an `mtime` value of 0 (zero).

Parameters:

hid_t `loc_id`	IN: File or group identifier. *Alternative:* An object identifier, `obj_id`
const char `*name`	IN: Name of the object for which status is being sought. *Alternative:* If the preceding parameter is the object's direct identifier, i.e., the `obj_id`, this parameter should be a dot (`.`).
hbool_t `follow_link`	IN: Link flag.
H5G_stat_t `*statbuf`	OUT: Buffer in which to return information about the object.

Returns:

Returns a non-negative value if successful, with the fields of `statbuf` (if non-null) initialized. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.1	Two new fields were added to the `H5G_stat_t` struct in this release.
1.8.0	Function deprecated in this release.

Name: H5Gget_objname_by_idx

Signature:

ssize_t `H5Gget_objname_by_idx`(*hid_t* `loc_id`, *hsize_t* `idx`, *char* `*name`, *size_t* `size`)

Purpose:

Returns a name of an object specified by an index.

Notice:

This function is deprecated in favor of the function `H5Lget_name_by_idx`.

Description:

`H5Gget_objname_by_idx` returns a name of the object specified by the index `idx` in the group `loc_id`.

The group is specified by a group identifier `loc_id`. If preferred, a file identifier may be passed in `loc_id`; that file's root group will be assumed.

`idx` is the transient index used to iterate through the objects in the group. The value of `idx` is any nonnegative number less than the total number of objects in the group, which is returned by the function `H5Gget_num_objs`. Note that this is a transient index; an object may have a different index each time a group is opened.

The object name is returned in the user-specified buffer `name`.

If the size of the provided buffer `name` is less or equal the actual object name length, the object name is truncated to `max_size - 1` characters.

Note that if the size of the object's name is unkown, a preliminary call to `H5Gget_objname_by_idx` with `name` set to NULL will return the length of the object's name. A second call to `H5Gget_objname_by_idx` can then be used to retrieve the actual name.

Parameters:

hid_t `loc_id`	IN: Group or file identifier.
hsize_t `idx`	IN: Transient index identifying object.
char `*name`	IN/OUT: Pointer to user-provided buffer the object name.
size_t `size`	IN: Name length.

Returns:

Returns the size of the object name if successful, or `0` if no name is associated with the group identifier. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.
1.8.0	Function deprecated in this release.

Name: H5Gget_objtype_by_idx

Signature:

int `H5Gget_objtype_by_idx`(*hid_t* `loc_id`, *hsize_t* `idx`)

Purpose:

Returns the type of an object specified by an index.

Notice:

This function is deprecated in favor of the function `H5Oget_info`.

Description:

`H5Gget_objtype_by_idx` returns the type of the object specified by the index `idx` in the group `loc_id`.

The group is specified by a group identifier `loc_id`. If preferred, a file identifier may be passed in `loc_id`; that file's root group will be assumed.

`idx` is the transient index used to iterate through the objects in the group. This parameter is described in more detail in the discussion of `H5Gget_objname_by_idx`.

The object type is returned as the function return value:

`H5G_LINK`	`0`	Object is a symbolic link.
`H5G_GROUP`	`1`	Object is a group.
`H5G_DATASET`	`2`	Object is a dataset.
`H5G_TYPE`	`3`	Object is a named datatype.

Parameters:

hid_t `loc_id`	IN: Group or file identifier.
hsize_t `idx`	IN: Transient index identifying object.

Returns:

Returns the type of the object if successful. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.
1.6.0	The function return type changed from *int* to the enumerated type *H5G_obj_t*.
1.8.0	Function deprecated in this release.

Name: H5Giterate

Signature:

int `H5Giterate`(*hid_t* `loc_id`, *const char* `*name`, *int* `*idx`, *H5G_iterate_t* `operator`, *void* `*operator_data`)

Purpose:

Iterates an operation over the entries of a group.

Notice:

This function is deprecated in favor of the function `H5Literate`.

Description:

`H5Giterate` iterates over the members of `name` in the file or group specified with `loc_id`. For each object in the group, the `operator_data` and some additional information, specified below, are passed to the `operator` function. The iteration begins with the `idx` object in the group and the next element to be processed by the operator is returned in `idx`. If `idx` is NULL, then the iterator starts at the first group member; since no stopping point is returned in this case, the iterator cannot be restarted if one of the calls to its operator returns non-zero. `H5Giterate` does not recursively follow links into subgroups of the specified group.

The prototype for `H5G_iterate_t` is:

`typedef` *herr_t* (*`H5G_iterate_t`) (*hid_t* `group_id`, *const char* * `member_name`, *void* *`operator_data`);

The operation receives the group identifier for the group being iterated over, `group_id`, the name of the current object within the group, `member_name`, and the pointer to the operator data passed in to `H5Giterate`, `operator_data`.

The return values from an operator are:

- ◊ Zero causes the iterator to continue, returning zero when all group members have been processed.
- ◊ Positive causes the iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the next group member.
- ◊ Negative causes the iterator to immediately return that value, indicating failure. The iterator can be restarted at the next group member.

`H5Giterate` assumes that the membership of the group identified by `name` remains unchanged through the iteration. If the membership changes during the iteration, the function's behavior is undefined.

`H5Giterate` is not recursive. In particular, if a member of `name` is found to be a group, call it `subgroup_a`, `H5Giterate` does not examine the members of `subgroup_a`. When recursive iteration is required, the application must handle the recursion, explicitly calling `H5Giterate` on discovered subgroups.

Parameters:

hid_t `loc_id`	IN: File or group identifier.
const char `*name`	IN: Group over which the iteration is performed.
int `*idx`	IN/OUT: Location at which to begin the iteration.
H5G_iterate_t `operator`	IN: Operation to be performed on an object at each step of the iteration.
void `*operator_data`	IN/OUT: Data associated with the operation.

Returns:

Returns the return value of the last operator if it was non-zero, or zero if all group members were processed. Otherwise returns a negative value.

Fortran90 Interface:

There is no direct FORTRAN couterpart for the C function `H5Giterate`. Instead, that functionality is provided by two FORTRAN functions:

`h5gn_members_f`	**Purpose:** Returns the number of group members.
`h5gget_obj_info_idx_f`	**Purpose:** Returns name and type of the group member identified by its index.

```
SUBROUTINE h5gn_members_f(loc_id, name, nmembers, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id        ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name        ! Name of the group
  INTEGER, INTENT(OUT) :: nmembers            ! Number of members in the group
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure
END SUBROUTINE h5gn_members_f


SUBROUTINE h5gget_obj_info_idx_f(loc_id, name, idx, &
                                 obj_name, obj_type, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id        ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name        ! Name of the group
  INTEGER, INTENT(IN) :: idx                  ! Index of member object
  CHARACTER(LEN=*), INTENT(OUT) :: obj_name   ! Name of the object
  INTEGER, INTENT(OUT) :: obj_type            ! Object type :
                                              !     H5G_LINK_F
                                              !     H5G_GROUP_F
                                              !     H5G_DATASET_F
                                              !     H5G_TYPE_F
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure
END SUBROUTINE h5gget_obj_info_idx_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Glink

Signature:

herr_t `H5Glink`(*hid_t* `loc_id`, *H5G_link_t* `link_type`, *const char* *`current_name`, *const char* *`new_name`)

Purpose:

Creates a link of the specified type from `new_name` to `current_name`.

Notice:

This function is deprecated in favor of the functions `H5Lcreate_hard` and `H5Lcreate_soft`.

Description:

`H5Glink` creates a new name for an object that has some current name, possibly one of many names it currently has.

If `link_type` is `H5G_LINK_HARD`, then `current_name` must specify the name of an existing object and both names are interpreted relative to `loc_id`, which is either a file identifier or a group identifier.

If `link_type` is `H5G_LINK_SOFT`, then `current_name` can be anything and is interpreted at lookup time relative to the group which contains the final component of `new_name`. For instance, if `current_name` is `./foo`, `new_name` is `./x/y/bar`, and a request is made for `./x/y/bar`, then the actual object looked up is `./x/y/./foo`.

Parameters:

hid_t `loc_id`	IN: File or group identifier.
H5G_link_t `link_type`	IN: Link type. Possible values are `H5G_LINK_HARD` and `H5G_LINK_SOFT`.
const char * `current_name`	IN: Name of the existing object if link is a hard link. Can be anything for the soft link.
const char * `new_name`	IN: New name for the object.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5glink_f

```
SUBROUTINE h5glink_f(loc_id, link_type, current_name, new_name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id      ! File or group location identifier
  INTEGER, INTENT(IN)        :: link_type   ! Link type, possible values are:
                                            !     H5G_LINK_HARD_F
                                            !     H5G_LINK_SOFT_F
  CHARACTER(LEN=*), INTENT(IN) :: current_name
                                            ! Current object name relative
                                            ! to loc_id
  CHARACTER(LEN=*), INTENT(IN) :: new_name  ! New object name
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure

END SUBROUTINE h5glink_f
```

History:

Release	C
1.8.0	Function deprecated in this release.

Name: H5Glink2

Signature:

herr_t `H5Glink2`(*hid_t* `curr_loc_id`, *const char* *`current_name`, *H5G_link_t* `link_type`, *hid_t* `new_loc_id`, *const char* *`new_name`)

Notice:

This function is deprecated in favor of the functions `H5Lcreate_hard` *and* `H5Lcreate_soft`*.*

Purpose:

Creates a link of the specified type from `current_name` to `new_name`.

Description:

`H5Glink2` creates a new name for an object that has some current name, possibly one of many names it currently has.

If `link_type` is `H5G_LINK_HARD`, then `current_name` must specify the name of an existing object. In this case, `current_name` and `new_name` are interpreted relative to `curr_loc_id` and `new_loc_id`, respectively, which are either file or group identifiers.

If `link_type` is `H5G_LINK_SOFT`, then `current_name` can be anything and is interpreted at lookup time relative to the group which contains the final component of `new_name`. For instance, if `current_name` is `./foo`, `new_name` is `./x/y/bar`, and a request is made for `./x/y/bar`, then the actual object looked up is `./x/y/./foo`.

Parameters:

hid_t `curr_loc_id`	IN: The file or group identifier for the original object.
*const char ** `current_name`	IN: Name of the existing object if link is a hard link. Can be anything for the soft link.
H5G_link_t `link_type`	IN: Link type. Possible values are `H5G_LINK_HARD` and `H5G_LINK_SOFT`.
hid_t `new_loc_id`	IN: The file or group identifier for the new link.
*const char ** `new_name`	IN: New name for the object.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5glink2_f

```
SUBROUTINE h5glink2_f(cur_loc_id, cur_name, link_type, new_loc_id, new_name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: cur_loc_id ! File or group location identifier
  CHARACTER(LEN=*), INTENT(IN) :: cur_name ! Name of the existing object
                                           ! is relative to cur_loc_id
                                           ! Can be anything for the soft link
  INTEGER, INTENT(IN) :: link_type         ! Link type, possible values are:
                                           !     H5G_LINK_HARD_F
                                           !     H5G_LINK_SOFT_F
  INTEGER(HID_T), INTENT(IN) :: new_loc_id ! New location identifier
  CHARACTER(LEN=*), INTENT(IN) :: new_name ! New object name
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure

END SUBROUTINE h5glink2_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Gmove

Signature:

herr_t `H5Gmove`(*hid_t* `loc_id`, *const char* `*src_name`, *const char* `*dst_name`)

Purpose:

Renames an object within an HDF5 file.

Notice:

This function is deprecated in favor of the function `H5Lmove`*.*

Description:

`H5Gmove` renames an object within an HDF5 file. The original name, `src_name`, is unlinked from the group graph and the new name, `dst_name`, is inserted as an atomic operation. Both names are interpreted relative to `loc_id`, which is either a file or a group identifier.

Warning:

Exercise care in moving groups as it is possible to render data in a file inaccessible with `H5Gmove`. See The Group Interface in the *HDF5 User's Guide*.

Parameters:

hid_t `loc_id`	IN: File or group identifier.
const char `*src_name`	IN: Object's original name.
const char `*dst_name`	IN: Object's new name.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gmove_f

```
SUBROUTINE h5gmove_f(loc_id, name, new_name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id     ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name     ! Original name of an object
  CHARACTER(LEN=*), INTENT(IN) :: new_name ! New name of an object
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5gmove_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Gmove2

Signature:

herr_t `H5Gmove2`(*hid_t* `src_loc_id`, *const char* `*src_name`, *hid_t* `dst_loc_id`, *const char* `*dst_name`)

Purpose:

Renames an object within an HDF5 file.

Notice:

This function is deprecated in favor of the function `H5Lmove`.

Description:

`H5Gmove2` renames an object within an HDF5 file. The original name, `src_name`, is unlinked from the group graph and the new name, `dst_name`, is inserted as an atomic operation.

`src_name` and `dst_name` are interpreted relative to `src_name` and `dst_name`, respectively, which are either file or group identifiers.

Warning:

Exercise care in moving groups as it is possible to render data in a file inaccessible with `H5Gmove`. See The Group Interface in the *HDF5 User's Guide*.

Parameters:

hid_t `src_loc_id`	IN: Original file or group identifier.
const char `*src_name`	IN: Object's original name.
hid_t `dst_loc_id`	IN: Destination file or group identifier.
const char `*dst_name`	IN: Object's new name.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gmove2_f

```
SUBROUTINE h5gmove2_f(src_loc_id, src_name, dst_loc_id, dst_name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: src_loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: src_name   ! Original name of an object
                                             ! relative to src_loc_id
  INTEGER(HID_T), INTENT(IN) :: dst_loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: dst_name   ! New name of an object
                                             ! relative to dst_loc_id
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5gmove2_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Gopen
Signature:

hid_t `H5Gopen(` *hid_t* `loc_id`, *const char* `*name`)

hid_t `H5Gopen(` *hid_t* `loc_id`, *const char* * `name`, *hid_t* `gapl_id`)

Purpose:

Opens an existing group in a file.

Description:

`H5Gopen` is a macro that is mapped to either `H5Gopen1` or `H5Gopen2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5” we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Gopen` is mapped to the most recent version of the function, currently `H5Gopen2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Gopen` will be mapped to `H5Gopen1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Gopen` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Gopen2`
Enable deprecated symbols	`H5Gopen2`
Disable deprecated symbols	`H5Gopen2`
Emulate Release 1.6 interface	`H5Gopen1`
<u>Function-level macros</u>	
`H5Gopen_vers = 2`	`H5Gopen2`
`H5Gopen_vers = 1`	`H5Gopen1`

Fortran90 Interface: h5gopen_f

```
SUBROUTINE h5gopen_f(loc_id, name, grp_id, hdferr, gapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the group
  INTEGER(HID_T), INTENT(OUT) :: grp_id  ! File identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: gapl_id
                                         ! Group access property list identifier
END SUBROUTINE h5gopen_f
```

History:

Release	**C**
1.8.0	The function `H5Gopen` renamed to `H5Gopen1` and deprecated in this release. The macro `H5Gopen` and the function `H5Gopen2` introduced in this release.

Name: H5Gopen1

Signature:

hid_t `H5Gopen1`(*hid_t* `loc_id`, *const char* *`name`)

Notice:

This function is deprecated in favor of the function `H5GOpen2` *or the macro* `H5GOpen`.

Purpose:

Opens an existing group for modification and returns a group identifier for that group.

Description:

`H5Gopen1` opens an existing group with the specified name at the specified location, `loc_id`.

The location is identified by a file or group identifier

`H5Gopen1` returns a group identifier for the group that was opened. This group identifier should be released by calling `H5Gclose` when it is no longer needed.

Parameters:

hid_t `loc_id`	IN: File or group identifier within which group is to be open.
*const char ** `name`	IN: Name of group to open.

Returns:

Returns a valid group identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Gopen`.

History:

Release	**C**
1.8.0	The function `H5Gopen` renamed to `H5Gopen1` and deprecated in this release.

Name: H5Gopen2

Signature:

hid_t `H5Gopen2`(*hid_t* `loc_id`, *const char* * `name`, *hid_t* `gapl_id`)

Purpose:

Opens an existing group with a group access property list.

Description:

`H5Gopen2` opens an existing group, `name`, at the location specified by `loc_id`.

With default settings, `H5Gopen2` provides similar functionality to that provided by `H5Gopen1`. The only difference is that `H5Gopen2` can provide a group access property list, `gapl_id`.

`H5Gopen2` returns a group identifier for the group that was opened. This group identifier should be released by calling `H5Gclose` when it is no longer needed.

Parameters:

hid_t `loc_id`	File or group identifier specifying the location of the group to be opened
const char `*name`	Name of the group to open
hid_t `gapl_id`	Group access property list identifier (No group access properties have been implemented at this time; use `H5P_DEFAULT`.)

Returns:

Returns a group identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Gopen`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Gset_comment

Signature:

herr_t `H5Gset_comment`(*hid_t* `loc_id`, *const char* *`name`, *const char* *`comment`)

Purpose:

Sets comment for specified object.

Notice:

This function is deprecated in favor of the function `H5Oset_comment`.

Description:

`H5Gset_comment` sets the comment for the object specified by `loc_id` and `name` to `comment`. Any previously existing comment is overwritten.

`loc_id` can specify any object in the file. `name` can be one of the following:
— The name of the object relative to `loc_id`
— An absolute name of the object, starting from `/`, the file's root group
— A dot (`.`), if `loc_id` fully specifies the object

If `comment` is the empty string or a null pointer, the comment message is removed from the object.

Comments should be relatively short, null-terminated, ASCII strings.

Comments can be attached to any object that has an object header, e.g., datasets, groups, and named datatypes, but not symbolic links.

Parameters:

hid_t `loc_id`	IN: Identifier of the file, group, dataset, or named datatype.
const char *`name`	IN: Name of the object whose comment is to be set or reset. `name` must be '.' (dot) if `loc_id` fully specifies the object for which the comment is to be set.
const char *`comment`	IN: The new comment.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gset_comment_f

```
SUBROUTINE h5gset_comment_f(loc_id, name, comment, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id      ! File, group, dataset, or
                                            ! named datatype identifier
  CHARACTER(LEN=*), INTENT(IN) :: name      ! Name of object
  CHARACTER(LEN=*), INTENT(IN) :: comment   ! Comment for the object
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5gset_comment_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

Name: H5Gunlink

Signature:

herr_t `H5Gunlink`(*hid_t* `loc_id`, *const char* *`name`)

Purpose:

Removes the link to an object from a group.

Notice:

This function is deprecated in favor of the function `H5Ldelete`.

Description:

`H5Gunlink` removes the object specified by `name` from the group graph and decrements the link count for the object to which `name` points. This action eliminates any association between `name` and the object to which `name` pointed.

Object headers keep track of how many hard links refer to an object; when the link count reaches zero, the object can be removed from the file. Objects which are open are not removed until all identifiers to the object are closed.

If the link count reaches zero, all file space associated with the object will be released, i.e., identified in memory as freespace. If any object identifier is open for the object, the space will not be released until after the object identifier is closed.

Note that space identified as freespace is available for re-use only as long as the file remains open; once a file has been closed, the HDF5 library loses track of freespace. See “Freespace Management” in the *HDF5 User's Guide* for further details.

Warning:

Exercise care in unlinking groups as it is possible to render data in a file inaccessible with `H5Gunlink`. See The Group Interface in the *HDF5 User's Guide*.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group containing the object.
const char * `name`	IN: Name of the object to unlink.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5gunlink_f

```
SUBROUTINE h5gunlink_f(loc_id, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the object to unlink
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5gunlink_f
```

History:

Release	**C**
1.8.0	Function deprecated in this release.

H5I: Identifier Interface

Identifier API Functions

These functions provides tools for working with object identifiers and object names.

The C Interface:

• H5Iget_file_id	• H5Iget_ref	• H5Itype_exists
• H5Iget_name	• H5Iinc_ref	• H5Iget_type_ref
• H5Iget_type	• H5Idec_ref	• H5Idec_type_ref
• H5Iobject_verify	• H5Iregister	• H5Iinc_type_ref
• H5Iremove_verify	• H5Iregister_type	• H5Iclear_type
• H5Isearch	• H5Idestroy_type	• H5Inmembers
• H5Iis_valid		

Alphabetical Listing

• H5Iclear_type	• H5Iget_ref	• H5Iobject_verify
• H5Idec_ref	• H5Iget_type	• H5Iregister
• H5Idec_type_ref	• H5Iget_type_ref	• H5Iregister_type
• H5Idestroy_type	• H5Iinc_ref	• H5Iremove_verify
• H5Iget_file_id	• H5Iinc_type_ref	• H5Isearch
• H5Iget_name	• H5Iis_valid	• H5Itype_exists
	• H5Inmembers	

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

• h5iget_name_f	• h5iget_ref_f	• h5idec_ref_f
• h5iget_type_f	• h5iinc_ref_f	• H5iis_valid_f

Name: H5Iclear_type

Signature:

herr_t `H5Iclear_type`(*H5I_type_t* `type`, *hbool_t* `force`)

Purpose:

Deletes all IDs of the given type

Description:

`H5Iclear_type` deletes all IDs of the type identified by the argument type.

The type s free function is first called on all of these IDs to free their memory, then they are removed from the type.

If the `force` flag is set to false, only those IDs whose reference counts are equal to 1 will be deleted, and all other IDs will be entirely unchanged. If the `force` flag is true, all IDs of this type will be deleted.

Parameters:

H5I_type_t `type`	IN: Identifier of ID type which is to be cleared of IDs
hbool_t `force`	IN: Whether or not to force deletion of all IDs

Returns:

Returns non-negative on success, negative on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Idec_ref

Signature:

int `H5Idec_ref(` *hid_t* `obj_id`)

Purpose:

Decrements the reference count for an object.

Description:

`H5Idec_ref` decrements the reference count of the object identified by `obj_id`.

The reference count for an object ID is attached to the information about an object in memory and has no relation to the number of links to an object on disk.

The reference count for a newly created object will be 1. Reference counts for objects may be explicitly modified with this function or with `H5Iinc_ref`. When an object ID's reference count reaches zero, the object will be closed. Calling an object ID's 'close' function decrements the reference count for the ID which normally closes the object, but if the reference count for the ID has been incremented with `H5Iinc_ref`, the object will only be closed when the reference count reaches zero with further calls to this function or the object ID's 'close' function.

If the object ID was created by a collective parallel call (such as `H5Dcreate`, `H5Gopen`, etc.), the reference count should be modified by all the processes which have copies of the ID. Generally this means that group, dataset, attribute, file and named datatype IDs should be modified by all the processes and that all other types of IDs are safe to modify by individual processes.

This function is of particular value when an application is maintaining multiple copies of an object ID. The object ID can be incremented when a copy is made. Each copy of the ID can then be safely closed or decremented and the HDF5 object will be closed when the reference count for that that object drops to zero.

Parameters:

hid_t `obj_id`	IN: Object identifier whose reference count will be modified.

Returns:

Returns a non-negative reference count of the object ID after decrementing it, if successful; otherwise a negative value is returned.

Fortran90 Interface: h5idec_ref_f

```
SUBROUTINE h5idec_ref_f(obj_id, ref_count, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id  !Object identifier
  INTEGER, INTENT(OUT) :: ref_count     !Reference count of object ID
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success, and -1 on failure
END SUBROUTINE h5idec_ref_f
```

History:

Release	**C**
1.6.2	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Idec_type_ref

Signature:

int `H5Idec_type_ref(` *H5I_type_t* `type`)

Purpose:

Decrements the reference count on an ID type.

Description:

`H5Idec_type_ref` decrements the reference count on an ID type. The reference count is used by the library to indicate when an ID type can be destroyed. If the reference count reaches zero, this function will destroy it.

The `type` parameter is the identifier for the ID type whose reference count is to be decremented. This identifier must have been created by a call to `H5Iregister_type`.

Parameters:

H5I_type_t `type`	IN: The identifier of the type whose reference count is to be decremented

Returns:

Returns the current reference count on success, negative on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Idestroy_type

Signature:

herr_t `H5Idestroy_type`(*H5I_type_t* `type`)

Purpose:

Removes the type `type` and all IDs within that type.

Description:

`H5Idestroy_type` deletes an entire ID type. All IDs of this type are destroyed and no new IDs of this type can be registered.

The type s free function is called on all of the IDs which are deleted by this function, freeing their memory. In addition, all memory used by this type s hash table is freed.

Since the *H5I_type_t* values of destroyed ID types are reused when new types are registered, it is a good idea to set the variable holding the value of the destroyed type to `H5I_UNINIT`.

Parameters:

H5I_type_t `type`	IN: Identifier of ID type which is to be destroyed

Returns:

Returns non-negative on success, negative on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Iget_file_id

Signature:

hid_t `H5Iget_file_id`(*hid_t* `obj_id`)

Purpose:

Retrieves an identifier for the file containing the specified object.

Description:

`H5Iget_file_id` returns the identifier of the file associated with the object referenced by `obj_id`.

`obj_id` can be a file, group, dataset, named datatype, or attribute identifier.

Note that the HDF5 Library permits an application to close a file while objects within the file remain open. If the file containing the object `obj_id` is still open, `H5Iget_file_id` will retrieve the existing file identifier. If there is no existing file identifier for the file, i.e., the file has been closed, `H5Iget_file_id` will reopen the file and return a new file identifier. In either case, the file identifier must eventually be released using `H5Fclose`.

Parameters:

hid_t `obj_id`	IN: Identifier of the object whose associated file identifier will be returned.

Returns:

Returns a file identifier on success, negative on failure.

Fortran90 Interface:

```
SUBROUTINE h5iget_file_id_f(obj_id, file_id, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: obj_id     ! Object identifier
  INTEGER(HID_T), INTENT(OUT) :: file_id    ! File identifier
  INTEGER, INTENT(OUT) :: hdferr            ! Error code

END SUBROUTINE h5iget_file_id_f
```

History:

Release	**C**
1.6.3	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Iget_name

Signature:

ssize_t `H5Iget_name`(*hid_t* `obj_id`, *char* `*name`, *size_t* `size`)

Purpose:

Retrieves a name of an object based on the object identifier.

Description:

`H5Iget_name` retrieves a name for the object identified by `obj_id`.

Up to `size` characters of the name are returned in `name`; additional characters, if any, are not returned to the user application.

If the length of the name, which determines the required value of `size`, is unknown, a preliminary `H5Iget_name` call can be made. The return value of this call will be the size in bytes of the object name. That value, plus 1 for a NULL terminator, is then assigned to `size` for a second `H5Iget_name` call, which will retrieve the actual name.

If the object identified by `obj_id` is an attribute, as determined via `H5Iget_type`, `H5Iget_name` retrieves the name of the object to which that attribute is attached. To retrieve the name of the attribute itself, use `H5Aget_name`.

If there is no name associated with the object identifier or if the name is `NULL`, `H5Iget_name` returns `0` (zero).

Note that an object in an HDF5 file may have multiple paths if there are multiple links pointing to it. This function may return any one of these paths. When possible, `H5Iget_name` returns the path with which the object was opened.

Parameters:

hid_t `obj_id`	IN: Identifier of the object. This identifier can refer to a group, dataset, or named datatype.
char `*name`	OUT: A name associated with the identifier.
size_t `size`	IN: The size of the `name` buffer; must be the size of the name in bytes plus 1 for a NULL terminator.

Returns:

Returns the length of the name if successful, returning `0` (zero) if no name is associated with the identifier. Otherwise returns a negative value.

Fortran90 Interface: h5iget_name_f

```
SUBROUTINE h5iget_name_f(obj_id, buf, buf_size, name_size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)    :: obj_id     ! Object identifier
  CHARACTER(LEN=*), INTENT(OUT) :: buf        ! Buffer to hold object name
  INTEGER(SIZE_T), INTENT(IN)   :: buf_size   ! Buffer size
  INTEGER(SIZE_T), INTENT(OUT)  :: name_size  ! Name size
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success, and -1 on failure
END SUBROUTINE h5iget_name_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Iget_ref

Signature:

int `H5Iget_ref(` *hid_t* `obj_id`)

Purpose:

Retrieves the reference count for an object.

Description:

`H5Iget_ref` retrieves the reference count of the object identified by `obj_id`.

The reference count for an object ID is attached to the information about an object in memory and has no relation to the number of links to an object on disk.

This function can also be used to check if an object ID is still valid. A non-negative return value from this function indicates that the ID is still valid.

Parameters:

hid_t `obj_id`	IN: Object identifier whose reference count will be retrieved.

Returns:

Returns a non-negative current reference count of the object ID if successful; otherwise a negative value is returned.

Fortran90 Interface: h5iget_ref_f

```
SUBROUTINE h5iget_ref_f(obj_id, ref_count, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id  !Object identifier
  INTEGER, INTENT(OUT) :: ref_count     !Reference count of object ID
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success, and -1 on failure
END SUBROUTINE h5iget_ref_f
```

History:

Release	**C**
1.6.2	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Iget_type

Signature:

H5I_type_t `H5Iget_type(` *hid_t* `obj_id`)

Purpose:

Retrieves the type of an object.

Description:

`H5Iget_type` retrieves the type of the object identified by `obj_id`.

Valid types returned by the function are

`H5I_FILE`	File
`H5I_GROUP`	Group
`H5I_DATATYPE`	Datatype
`H5I_DATASPACE`	Dataspace
`H5I_DATASET`	Dataset
`H5I_ATTR`	Attribute

If no valid type can be determined or the identifier submitted is invalid, the function returns

`H5I_BADID`	Invalid identifier

This function is of particular value in determining the type of object closing function (`H5Dclose`, `H5Gclose`, etc.) to call after a call to `H5Rdereference`.

Parameters:

hid_t `obj_id`	IN: Object identifier whose type is to be determined.

Returns:

Returns the object type if successful; otherwise `H5I_BADID`.

Fortran90 Interface: h5iget_type_f

```
SUBROUTINE h5iget_type_f(obj_id, type, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id  !Object identifier
  INTEGER, INTENT(OUT) :: type          !type of an object.
                                        !possible values are:
                                        !H5I_FILE_F
                                        !H5I_GROUP_F
                                        !H5I_DATATYPE_F
                                        !H5I_DATASPACE_F
                                        !H5I_DATASET_F
                                        !H5I_ATTR_F
                                        !H5I_BADID_F
  INTEGER, INTENT(OUT) :: hdferr        ! E rror code
                                        ! 0 on success, and -1 on failure
END SUBROUTINE h5iget_type_f
```

Name: H5Iget_type_ref

Signature:

int `H5Iget_type_ref(` *H5I_type_t* `type`)

Purpose:

Retrieves the reference count on an ID type.

Description:

`H5Iget_type_ref` retrieves the reference count on an ID type. The reference count is used by the library to indicate when an ID type can be destroyed.

The `type` parameter is the identifier for the ID type whose reference count is to be retrieved. This identifier must have been created by a call to `H5Iregister_type`.

Parameters:

H5I_type_t `type`	IN: The identifier of the type whose reference count is to be retrieved

Returns:

Returns the current reference count on success, negative on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Iinc_ref

Signature:

int `H5Iinc_ref(` *hid_t* `obj_id`)

Purpose:

Increments the reference count for an object.

Description:

`H5Iinc_ref` increments the reference count of the object identified by `obj_id`.

The reference count for an object ID is attached to the information about an object in memory and has no relation to the number of links to an object on disk.

The reference count for a newly created object will be 1. Reference counts for objects may be explicitly modified with this function or with `H5Idec_ref`. When an object ID's reference count reaches zero, the object will be closed. Calling an object ID's 'close' function decrements the reference count for the ID which normally closes the object, but if the reference count for the ID has been incremented with this function, the object will only be closed when the reference count reaches zero with further calls to `H5Idec_ref` or the object ID's 'close' function.

If the object ID was created by a collective parallel call (such as `H5Dcreate`, `H5Gopen`, etc.), the reference count should be modified by all the processes which have copies of the ID. Generally this means that group, dataset, attribute, file and named datatype IDs should be modified by all the processes and that all other types of IDs are safe to modify by individual processes.

This function is of particular value when an application is maintaining multiple copies of an object ID. The object ID can be incremented when a copy is made. Each copy of the ID can then be safely closed or decremented and the HDF5 object will be closed when the reference count for that that object drops to zero.

Parameters:

hid_t `obj_id`	IN: Object identifier whose reference count will be modified.

Returns:

Returns a non-negative reference count of the object ID after incrementing it if successful; otherwise a negative value is returned.

Fortran90 Interface: h5iinc_ref_f

```
SUBROUTINE h5iinc_ref_f(obj_id, ref_count, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id  !Object identifier
  INTEGER, INTENT(OUT) :: ref_count     !Reference count of object ID
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success, and -1 on failure
END SUBROUTINE h5iinc_ref_f
```

History:

Release	**C**
1.6.2	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Iinc_type_ref
Signature:
int `H5Iinc_type_ref(` *H5I_type_t* `type`)
Purpose:
Increments the reference count on an ID type.
Description:
`H5Iinc_type_ref` increments the reference count on an ID type. The reference count is used by the library to indicate when an ID type can be destroyed.

The `type` parameter is the identifier for the ID type whose reference count is to be incremented. This identifier must have been created by a call to `H5Iregister_type`.
Parameters:

H5I_type_t `type`	IN: The identifier of the type whose reference count is to be incremented

Returns:
Returns the current reference count on success, negative on failure.
Fortran90 Interface:
This function is not supported in FORTRAN 90.

Name: H5Inmembers

Signature:

herr_t `H5Inmembers`(*H5I_type_t* `type`, *hsize_t* *`num_members`)

Purpose:

Returns the number of identifiers in a given identifier type.

Description:

`H5Inmembers` returns the number of identifiers of the identifier type specified in `type`.

The number of identifiers is returned in `num_members`. If no identifiers of this type have been registered, the type does not exist, or it has been destroyed, `num_members` is returned with the value `0`.

Parameters:

H5I_type_t `type`	IN: Identifier for the identifier type whose member count will be retrieved
hsize_t *`num_members`	OUT: Number of identifiers of the specified identifier type.

Returns:

Returns a non-negative value on success; otherwise returns negative value.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Iobject_verify

Signature:

void * `H5Iobject_verify`(*hid_t* `id`, *H5I_type_t* `id_type`)

Purpose:

Returns the object referenced by id.

Description:

`H5Iobject_verify` returns a pointer to the memory referenced by `id` after verifying that `id` is of type `id_type`. This function is analogous to dereferencing a pointer in C with type checking.

`H5Iregister`(*H5I_type_t* `type`, *void* *`object`) takes an *H5I_type_t* and a void pointer to an object, returning an *hid_t* of that type. This *hid_t* can then be passed to `H5Iobject_verify` along with its type to retrieve the object.

`H5Iobject_verify` does not change the ID it is called on in any way (as opposed to `H5Iremove_verify`, which removes the ID from its type s hash table).

Parameters:

hid_t `id`	IN: ID to be dereferenced
H5I_type_t `type`	IN: ID type to which id should belong

Returns:

Pointer to the object referenced by id on success, `NULL` on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Iregister

Signature:

hid_t `H5Iregister(` *H5I_type_t* `type`, *void* `*object`)

Purpose:

Creates and returns a new ID.

Description:

`H5Iregister` allocates space for a new ID and returns an identifier for it.

The `type` parameter is the identifier for the ID type to which this new ID will belong. This identifier must have been created by a call to `H5Iregister_type`.

The `object` parameter is a pointer to the memory which the new ID will be a reference to. This pointer will be stored by the library and returned to you via a call to `H5Iobject_verify`.

Parameters:

H5I_type_t `type`	IN: The identifier of the type to which the new ID will belong
void `*object`	IN: Pointer to memory for the library to store

Returns:

Returns the new ID on success, negative on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Iregister_type

Signature:

H5I_type_t `H5Iregister_type`(*size_t* `hash_size`, *unsigned* `reserved`, *H5I_free_t* `free_func`)

Purpose:

Creates and returns a new ID type.

Description:

`H5Iregister_type` allocates space for a new ID type and returns an identifier for it.

The `hash_size` parameter indicates the minimum size of the hash table used to store IDs in the new type.

The `reserved` parameter indicates the number of IDs in this new type to be reserved. Reserved IDs are valid IDs which are not associated with any storage within the library.

The `free_func` parameter is a function pointer to a function which returns an *herr_t* and accepts a *void **. The purpose of this function is to deallocate memory for a single ID. It will be called by `H5Iclear_type` and `H5Idestroy_type` on each ID. This function is NOT called by `H5Iremove_verify`. The *void ** will be the same pointer which was passed in to the `H5Iregister` function. The `free_func` function should return 0 on success and -1 on failure.

Parameters:

size_t `hash_size`	IN: Size of the hash table (in entries) used to store IDs for the new type
unsigned `reserved`	IN: Number of reserved IDs for the new type
H5I_free_t `free_func`	IN: Function used to deallocate space for a single ID

Returns:

Returns the type identifier on success, negative on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Iremove_verify

Signature:

void *`H5Iremove_verify`(*hid_t* `id`, *H5I_type_t* `id_type`)

Purpose:

Removes an ID from internal storage.

Description:

`H5Iremove_verify` first ensures that `id` belongs to `id_type`. If so, it removes `id` from internal storage and returns the pointer to the memory it referred to. This pointer is the same pointer that was placed in storage by `H5Iregister`. If `id` does not belong to `id_type`, then `NULL` is returned.

The `id` parameter is the ID which is to be removed from internal storage. Note: this function does NOT deallocate the memory that `id` refers to. The pointer returned by `H5Iregister` must be deallocated by the user to avoid memory leaks.

The `type` parameter is the identifier for the ID type which `id` is supposed to belong to. This identifier must have been created by a call to `H5Iregister_type`.

Parameters:

hid_t `id`	IN: The ID to be removed from internal storage
H5I_type_t `type`	IN: The identifier of the type whose reference count is to be retrieved

Returns:

Returns a pointer to the memory referred to by `id` on success, `NULL` on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Isearch
Signature:

void *`H5Isearch`(*H5I_type_t* `type`, *H5I_search_func_t* `func`, *void* *`key`)

Purpose:

Finds the memory referred to by an ID within the given ID type such that some criterion is satisfied.

Description:

`H5Isearch` searches through a give ID type to find an object that satisfies the criteria defined by `func`. If such an object is found, the pointer to the memory containing this object is returned. Otherwise, `NULL` is returned. To do this, `func` is called on every member of `type`. The first member to satisfy `func` is returned.

The `type` parameter is the identifier for the ID type which is to be searched. This identifier must have been created by a call to `H5Iregister_type`.

The parameter `func` is a function pointer to a function which takes three parameters. The first parameter is a *void **. It will be a pointer the object to be tested. This is the same object that was placed in storage using `H5Iregister`. The second parameter is a *hid_t*. It is the ID of the object to be tested. The last parameter is a *void **. This is the `key` parameter and can be used however the user finds helpful. Or it can simply be ignored if it is not needed. `func` returns 0 if the object it is testing does not pass its criteria. A non-zero value should be returned if the object does pass its criteria.

The `key` parameter will be passed to the search function as a parameter. It can be used to further define the search at run-time.

Parameters:

H5I_type_t `type`	IN: The identifier of the type to be searched
H5I_search_func_t `func`	IN: The function defining the search criteria
void *`key`	IN: A key for the search function

Returns:

Returns a pointer to the object which satisfies the search function on success, `NULL` on failure.

Fortran90 Interface:

This function is not supported in FORTRAN 90.

Name: H5Itype_exists

Signature:

htri_t `H5Itype_exists(` *H5I_type_t* `type`)

Purpose:

Determines whether an identifier type is registered.

Description:

`H5Itype_exists` determines whether the given identifier type, `type`, is registered with the library.

Parameters:

H5I_type_t `type`	IN: Identifier type.

Returns:

Returns `1` if the type is registered and `0` if not. Returns a negative value on failure.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

H5L: Link Interface

Link API Functions

The Link interface, H5L, functions create and manipulate links in an HDF5 group. This interface includes functions that enable the creation and use of user-defined link classes.

The C Interfaces:

- H5Lcreate_hard
- H5Lcreate_soft
- H5Lcreate_external
- H5Lexists
- H5Lmove
- H5Lcopy
- H5Ldelete

- H5Lget_info
- H5Lget_val
- H5Lunpack_elink_val

- H5Lcreate_ud
- H5Lregister
- H5Lunregister
- H5Lis_registered

- H5Literate
- H5Literate_by_name
- H5Lvisit
- H5Lvisit_by_name
- H5Lget_info_by_idx
- H5Lget_name_by_idx
- H5Lget_val_by_idx
- H5Ldelete_by_idx

Alphabetical Listing

- H5Lcopy
- H5Lcreate_external
- H5Lcreate_hard
- H5Lcreate_soft
- H5Lcreate_ud
- H5Ldelete
- H5Ldelete_by_idx
- H5Lexists
- H5Lget_info
- H5Lget_info_by_idx
- H5Lget_name_by_idx
- H5Lget_val
- H5Lget_val_by_idx
- H5Lis_registered
- H5Literate
- H5Literate_by_name
- H5Lmove
- H5Lregister
- H5Lunpack_elink_val
- H5Lunregister
- H5Lvisit
- H5Lvisit_by_name

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5lcopy_f
- h5lcreate_external_f
- h5lcreate_hard_f
- h5lcreate_soft_f
- h5ldelete_f
- h5ldelete_by_idx_f
- h5lexists_f
- h5lget_info_f
- h5lget_info_by_idx_f
- h5lget_name_by_idx_f
- h5lis_registered_f
- h5lmove_f

Name: H5Lcopy

Signature:

herr_t `H5Lcopy`(*hid_t* `src_loc_id`, *const char* *`src_name`, *hid_t* `dest_loc_id`, *const char* *`dest_name`, *hid_t* `lcpl_id` *hid_t* `lapl_id`)

Purpose:

Copies a link from one location to another.

Description:

`H5Lcopy` copies the link specified by `src_name` from the file or group specified by `src_loc_id` to the destination location `dest_loc_id`. The new copy of the link is created with the name `dest_name`.

The destination location, as specified in `dest_loc_id`, must be either the current file or a group in the current file. If `dest_loc_id` is the file identifier, the copy is placed in the file's root group.

The new link is created with the creation and access property lists specified by `lcpl_id` and `lapl_id`. The interpretation of `lcpl_id` is limited in the manner described in the next paragraph.

`H5Lcopy` retains the creation time and the target of the original link. However, since the link may be renamed, the character encoding is that specified in `lcpl_id` rather than that of the original link. Other link creation properties are ignored.

If the link is a soft link, also known as a symbolic link, its target is interpreted relative to the location of the copy.

Several properties are available to govern the behavior of `H5Lcopy`. These properties are set in the link creation and access property lists, `lcpl_id` and `lapl_id`, respectively. The property controlling creation of missing intermediate groups is set in the link creation property list with `H5Pset_create_intermediate_group`; this function ignores any other properties in the link creation property list. Properties controlling character encoding, link traversals, and external link prefixes are set in the link access property list with `H5Pset_char_encoding`, `H5Pset_nlinks`, and `H5Pset_elink_prefix`.

`H5Lcopy` does not affect the object that the link points to.

`H5Lcopy` cannot copy links to a different file; to copy objects to a different file, see `H5Ocopy`.

Parameters:

hid_t `src_loc_id`	IN: Location identifier of the source link
const char *`src_name`	IN: Name of the link to be copied
hid_t `dest_loc_id`	IN: Location identifier specifying the destination of the copy
const char *`dest_name`	IN: Name to be assigned to the new copy
hid_t `lcpl_id`	IN: Link creation property list identifier
hid_t `lapl_id`	IN: Link access property list identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5lcopy_f

```
SUBROUTINE h5lcopy_f(src_loc_id, src_name, dest_loc_id, dest_name, hdferr, &
                     lcpl_id, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: src_loc_id
                                   ! Location identifier of the source link
  CHARACTER(LEN=*), INTENT(IN) :: src_name
                                   ! Name of the link to be copied
  INTEGER(HID_T), INTENT(IN) :: dest_loc_id
                                   ! Location identifier specifying the
                                   ! destination of the copy
  CHARACTER(LEN=*), INTENT(IN) :: dest_name
                                   ! Name to be assigned to the new copy
  INTEGER, INTENT(OUT) :: hdferr   ! Error code:
                                   ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                                   ! Link creation property list identifier
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                   ! Link access property list identifier
END SUBROUTINE h5lcopy_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lcreate_external

Signature:

herr_t `H5Lcreate_external`(*const char* *`file_name`, *const char* *`object_name`, *hid_t* `link_loc_id`, *const char* *`link_name`, *hid_t* `lcpl_id`, *hid_t* `lapl_id`)

Purpose:

Creates a soft link to an object in a different file.

Description:

`H5Lcreate_external` creates a new soft link to an external object, which is an object in a different HDF5 file from the location of the link.

`file_name` identifies the target file containing the target object; `object_name` specifies the path of the target object within that file. `object_name` must start at the target file's root group but is not interpreted until lookup time.

`link_loc_id` and `link_name` specify the location and name, respectively, of the new link. `link_name` is interpreted relative to `link_loc_id`

`lcpl_id` is the link creation property list used in creating the new link.

`lapl_id` is the link access property list used in traversing to the place where the new link is to be created.

An external link behaves similarly to a soft link, and like a soft link in an HDF5 file, it may *dangle*: the target file and object need not exist at the time that the external link is created.

When the external link `link_name` is accessed, the library will search for the target file `file_name` as described below:

◊ If `file_name` is a relative pathname, the following steps are performed:
- · The library will get the prefix(es) set in the environment variable HDF5_EXT_PREFIX and will try prepending each prefix to `file_name` to form a new `file_name`.
- · If the new `file_name` does not exist or HDF5_EXT_PREFIX is not set, the library will get the prefix set via `H5Pset_elink_prefix` and prepend it to `file_name` to form a new `file_name`.
- · If the new `file_name` does not exist or no prefix is being set by `H5Pset_elink_prefix`, then the path of the file associated with `link_loc_id` is obtained. This path can be the absolute path or the current working directory+relative path of that file when it is created/opened. The library will prepend this path to `file_name` to form a new `file_name`.
- · If the new `file_name` does not exist, then the library will look for `file_name` and will return failure/success accordingly.

◊ If `file_name` is an absolute pathname, the library will try to find `file_name`. If `file_name` does not exist, the `file_name` is stripped of the directory paths to form a new `file_name`. See examples below on how `file_name` is stripped to form a new `file_name`. The search for the new `file_name` then follows the same steps as described above for a relative pathname.

Note that `file_name` is considered to be an absolute pathname when the following condition is true:

◊ For Unix, the first character of `file_name` is a '/' , e.g. "/tmp/A.h5".

If `file_name` does not exist, the new `file_name` after stripping `file_name` will be "A.h5".

◊ For Windows, there are 3 cases:

1. `file_name` is an absolute drive with absolute pathname, e.g. "C:\tmp\A.h5".

   If `file_name` does not exist, the new `file_name` after stripping `file_name` will be "A.h5".
2. `file_name` is an absolute pathname without specifying drive name, e.g. "\tmp\A.h5".

   If `file_name` does not exist, the new `file_name` after stripping `file_name` will be "A.h5".
3. `file_name` is an absolute drive with relative pathname, e.g. "C:tmp\A.h5".

   If `file_name` does not exist, the new `file_name` after stripping `file_name` will be "tmp\A.h5".

The library opens the target file `file_name` with the file access property list that is set via `H5Pset_elink_fapl()` when the external link `link_name` is accessed. If no such identifier is set, the library uses the file access property list identifier of the file associated with `link_loc_id` to open the target file.

Parameters:

const char * `file_name`	IN: Name of the target file containing the target object.
const char *`object_name`	IN: Path within the target file to the target object.
hid_t `link_loc_id`	IN: The file or group identifier for the new link.
const char * `link_name`	IN: The name of the new link.
hid_t `lcpl_id`	IN: Link creation property list identifier.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5lcreate_external_f

```
SUBROUTINE h5lcreate_external_f(file_name, obj_name, link_loc_id, link_name, &
                                hdferr, lcpl_id, lapl_id)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: file_name
                      ! Name of the file containing the target object. Neither
                      ! the file nor the target object is required to exist.
                      ! May be the file the link is being created in.
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                      ! Name of the target object, which must already exist.
  INTEGER(HID_T), INTENT(IN) :: link_loc_id
                      ! The file or group identifier for the new link.
  CHARACTER(LEN=*), INTENT(IN) :: link_name
                      ! The name of the new link.
  INTEGER, INTENT(OUT) :: hdferr
                      ! Error code:
                      ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                      ! Link creation property list identifier.
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                      ! Link access property list identifier.
END SUBROUTINE h5lcreate_external_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lcreate_hard

Signature:

herr_t `H5Lcreate_hard`(*hid_t* `obj_loc_id`, *const char* *`obj_name`, *hid_t* `link_loc_id`, *const char* *`link_name`, *hid_t* `lcpl_id`, *hid_t* `lapl_id`)

Purpose:

Creates a hard link to an object.

Description:

`H5Lcreate_hard` creates a new hard link to a pre-existing object in an HDF5 file. The new link may be one of many that point to that object.

`obj_loc_id` and `obj_name` specify the location and name, respectively, of the target object, i.e., the object that the new hard link points to.

`link_loc_id` and `link_name` specify the location and name, respectively, of the new hard link.

`obj_name` and `link_name` are interpreted relative to `obj_loc_id` and `link_loc_id`, respectively.

The target object must already exist in the file.

`lcpl_id` and `lapl_id` are the link creation and access property lists associated with the new link.

Hard and soft links are for use only if the target object is in the current file. If the desired target object is in a different file from the new link, an external link may be created with `H5Lcreate_external`.

The HDF5 library keeps a count of all hard links pointing to an object; if the hard link count reaches zero (`0`), the object will be deleted from the file. Creating new hard links to an object will prevent it from being deleted if other links are removed. The library maintains no similar count for soft links and they can dangle.

Parameters:

hid_t `obj_loc_id`	IN: The file or group identifier for the target object.
const char *`obj_name`	IN: Name of the target object, which must already exist.
hid_t `link_loc_id`	IN: The file or group identifier for the new link.
const char * `link_name`	IN: The name of the new link.
hid_t `lcpl_id`	IN: Link creation property list identifier.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5lcreate_hard_f

```
SUBROUTINE h5lcreate_hard_f(obj_loc_id, obj_name, link_loc_id, link_name, &
                            hdferr, lcpl_id, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_loc_id
                          ! The file or group identifier for the target object.
  CHARACTER(LEN=*), INTENT(IN) :: obj_name
                          ! Name of the target object, which must already exist.
  INTEGER(HID_T), INTENT(IN) :: link_loc_id
                          ! The file or group identifier for the new link.
  CHARACTER(LEN=*), INTENT(IN) :: link_name
                          ! The name of the new link.
  INTEGER, INTENT(OUT) :: hdferr
                          ! Error code:
                          ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) ::   lcpl_id
                          ! Link creation property list identifier.
  INTEGER(HID_T), OPTIONAL, INTENT(IN) ::   lapl_id
                          ! Link access property list identifier.
END SUBROUTINE h5lcreate_hard_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lcreate_soft

Signature:

herr_t `H5Lcreate_soft(` *const char* `*target_path`, *hid_t* `link_loc_id`, *const char* `*link_name`, *hid_t* `lcpl_id`, *hid_t* `lapl_id`)

Purpose:

Creates a soft link to an object.

Description:

`H5Lcreate_soft` creates a new soft link to an object in an HDF5 file. The new link may be one of many that point to that object.

`target_path` specifies the path to the target object, i.e., the object that the new soft link points to. `target_path` can be anything and is interpreted at lookup time. This path may be absolute in the file or relative to `link_loc_id`.

`link_loc_id` and `link_name` specify the location and name, respectively, of the new soft link. `link_name` is interpreted relative to `link_loc_id`

`lcpl_id` and `lapl_id` are the link creation and access property lists associated with the new link.

For instance, if `target_path` is `./foo`, `link_loc_id` specifies `./x/y/bar`, and the name of the new link is `new_link`, then a subsequent request for `new_link` will look up the object `./x/y/bar/foo`.

`H5Lcreate_soft` is for use only if the target object is in the current file. If the desired target object is in a different file from the new link, use `H5Lcreate_external` to create an external link.

Soft links and external links are also known as symbolic links as they use a name to point to an object; hard links employ an object's address in the file.

Unlike hard links, a soft link in an HDF5 file is allowed to *dangle*, meaning that the target object need not exist at the time that the link is created.

The HDF5 library does not keep a count of soft links as it does of hard links.

Parameters:

const char `*target_path`	IN: Path to the target object, which is not required to exist.
hid_t `link_loc_id`	IN: The file or group identifier for the new link.
const char `* link_name`	IN: The name of the new link.
hid_t `lcpl_id`	IN: Link creation property list identifier.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5lcreate_soft_f

```
SUBROUTINE h5lcreate_soft_f(target_path, link_loc_id, link_name, hdferr, &
                            lcpl_id, lapl_id)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: target_path
                                 ! Path to the target object,
                                 ! which is not required to exist.
  INTEGER(HID_T), INTENT(IN) :: link_loc_id
                                 ! The file or group identifier for the new link.
  CHARACTER(LEN=*), INTENT(IN) :: link_name
                                 ! The name of the new link.
  INTEGER, INTENT(OUT) :: hdferr ! Error code:
                                 ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                                 ! Link creation property list identifier.
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                 ! Link access property list identifier.
END SUBROUTINE h5lcreate_soft_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lcreate_ud

Signature:

herr_t `H5Lcreate_ud`(*hid_t* `link_loc_id`, *const char* *`link_name`, *H5L_type_t* `link_type`, *const char* *`udata`, *size_t* `udata_size`, *hid_t* `lcpl_id`, *hid_t* `lapl_id`)

Purpose:

Creates a link of a user-defined type.

Description:

`H5Lcreate_ud` creates a link of user-defined type `link_type` named `link_name` at the location specified in `link_loc_id` with user-specified data `udata`.

`link_name` is interpreted relative to `link_loc_id`.

The link class of the new link, `link_type`, must already be registered with the library. Such user-defined link classes are registered with `H5Lregister`.

The `H5L_type_t`, the ENUM specifying valid link classes, is defined in `H5Lpublic.h`. Valid values without registered user-defined link types include the following:

```
H5L_LINK_HARD
H5L_LINK_SOFT
H5L_LINK_EXTERNAL
```

Registering a user-defined link type with `H5Lregister` adds a user-defined symbol to the `H5L_type_t` ENUM; that user-defined symbol is then passed in the `link_type` parameter of `H5Lcreate_ud` and will be returned by `H5Lget_info`.

The format of the information pointed to by `udata` is defined by the user. `udata_size` specifies the size of the `udata` buffer. `udata` may be NULL if `udata_size` is zero (`0`).

The property lists specified by `lcpl_id` and `lapl_id` specify properties used to create and access the link.

Note:

Though the external link type is included in the HDF5 Library distribution, it is implemented as a user-defined link type. This was done, in part, to provide a model for implementation of other user-defined links.

Parameters:

hid_t `link_loc_id`	IN: Link location identifier
const char *`link_name`	IN: Link name
H5L_type_t `link_type`	IN: User-defined link class
const char *`udata`	IN: User-supplied link information
size_t `udata_size`	IN: Size of `udata` buffer
hid_t `lcpl_id`	IN: Link creation property list identifier
hid_t `lapl_id`	IN: Link access property list identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Last modified: 10 April 2009

Name: H5Lexists

Signature:

htri_t `H5Lexists`(*hid_t* `loc_id`, *const char* `*name`, *hid_t* `lapl_id`)

Purpose:

Determine whether a link with the specified name exists in a group.

Description:

`H5Lexists` allows an application to determine whether the link `name` exists in the group or file specified with `loc_id`. The link may be of any type; only the presence of a link with that name is checked.

Note that `H5Lexists` verifies only that the target link exists. If `name` includes either a relative path or an absloute path to the target link, intermediate steps along the path must be verified before the existence of the target link can be safely checked. If the path is not verified and an intermediate element of the path does not exist, `H5Lexists` will fail. The example in the next paragraph illustrates one step-by-step method for verifying the existence of a link with a relative or absolute path.

Example: Use the following steps to verify the existence of the link `datasetD` in the group `group1/group2/softlink_to_group3/`, where `group1` is a member of the group specified by `loc_id`:

◊ First use `H5Lexists` to verify that `group1` exists.
◊ If `group1` exists, use `H5Lexists` again, this time with `name` set to `group1/group2`, to verify that`group2` exists.
◊ If `group2` exists, use `H5Lexists` with `name` set to `group1/group2/softlink_to_group3` to verify that `softlink_to_group3` exists.

◊ If `softlink_to_group3` exists, you can now safely use `H5Lexists` with `name` set to `group1/group2/softlink_to_group3/datasetD` to verify that the target link, `datasetD`, exists.

If the link to be verified is specified with an absolute path, the same approach should be used, but starting with the first link in the file's root group. For instance, if `datasetD` were in `/group1/group2/softlink_to_group3`, the first call to `H5Lexists` would have `name` set to `/group1`.

Note that this is an outline and does not include all necessary details. Depending on circumstances, for example, you may need to verify that an intermediate link points to a group and that a soft link points to an existing target.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group to query.
const char `*name`	IN: The name of the link to check.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns `TRUE` or `FALSE` if successful; otherwise returns a negative value.

Failure Modes:

`H5Lexists` checks the existence of only the final element in a relative or absolute path; it does not check any other path elements. The function will therefore fail when both of the following conditions exist:

- ◊ `name` is not local to the group specified by `loc_id` or, if `loc_id` is something other than a group identifier, `name` is not local to the root group.
- ◊ Any element of the relative path or absolute path in `name`, except the target link, does not exist.

Fortran90 Interface: h5lexists_f

```
SUBROUTINE h5lexists_f(loc_id, name, link_exists, hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id ! Identifier of file or group to query.
  CHARACTER(LEN=*), INTENT(IN) :: name ! Link name to check.
  LOGICAL, INTENT(OUT) :: link_exists  ! .TRUE. if exists, .FALSE. otherwise
  INTEGER, INTENT(OUT) :: hdferr       ! Error code:
                                       ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                       ! Link access property list identifier.
END SUBROUTINE h5lexists_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lget_info

Signature:

herr_t `H5Lget_info`(*hid_t* `link_loc_id`, *const char* *`link_name`, *H5L_info_t* *`link_buff`, *hid_t* `lapl_id`)

Purpose:

Returns information about a link.

Description:

`H5Lget_info` returns information about the specified link through the `link_buff` argument.

A file or group identifier, `link_loc_id`, specifies the location of the link. A link name, `link_name`, interpreted relative to `loc_id`, specifies the link being queried.

`lapl_id` is the link access property list associated with the link `link_name`. In the general case, when default link access properties are acceptable, this can be passed in as `H5P_DEFAULT`. An example of a situation that requires a non-default link access property list is when the link is an external link; an external link may require that a link prefix be set in a link access property list (see `H5Pset_elink_prefix`).

`H5Lget_info` returns information about `link_name` in the data structure `H5L_info_t`, which is described below and defined in `H5Lpublic.h`. This structure is returned in the buffer `link_buff`.

```
typedef struct {
    H5L_type_t     type;
    hbool_t        corder_valid;
    int64_t        corder;
    H5T_cset_t     cset;
    union {
        haddr_t    address;
        size_t     val_size;
    } u;
} H5L_info_t;
```

In the above struct, `type` specifies the link class. Valid values include the following:

`H5L_LINK_HARD`	Hard link
`H5L_LINK_SOFT`	Soft link
`H5L_LINK_EXTERNAL`	External link
`H5L_LINK_ERROR`	Error

There will be additional valid values if user-defined links have been registered.

`corder` specifies the link's creation order position while `corder_valid` indicates whether the value in `corder` is valid.

If `corder_valid` is `TRUE`, the value in `corder` is known to be valid; if `corder_valid` is `FALSE`, the value in `corder` is presumed to be invalid;

`corder` starts at zero (`0`) and is incremented by one (`1`) as new links are created. But higher-numbered entries are not adjusted when a lower-numbered link is deleted; the deleted link's creation order position is simply left vacant. In such situations, the value of `corder` for the last link created will be larger than the number of links remaining in the group.

`cset` specifies the character set in which the link name is encoded. Valid values include the following:

`H5T_CSET_ASCII`	US ASCII
`H5T_CSET_UTF8`	UTF-8 Unicode encoding

`address` and `val_size` are returned for hard and symbolic links, respectively. Symbolic links include soft and external links and some user-defined links.

If the link is a hard link, `address` specifies the file address that the link points to.

If the link is a symbolic link, `val_size` will be the length of the link value, e.g., the length of the name of the pointed-to object with a null terminator.

Parameters:

hid_t `link_loc_id`	IN: File or group identifier.
const char `*link_name`	IN: Name of the link for which information is being sought.
H5L_info_t `*link_buff`	OUT: Buffer in which link information is returned.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns a non-negative value if successful, with the fields of `link_buff` (if non-null) initialized. Otherwise returns a negative value.

Fortran90 Interface: h5lget_info_f

```
SUBROUTINE h5lget_info_f(link_loc_id, link_name, &
     cset, corder, f_corder_valid, link_type, address, val_size, &
     hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: link_loc_id
                        ! File or group identifier.
  CHARACTER(LEN=*), INTENT(IN) :: link_name
                        ! Name of the link for which information is being sought.
  INTEGER, INTENT(OUT) :: cset
                        ! Indicates the character set used for the link’s name.
  INTEGER, INTENT(OUT) :: corder
                        ! Specifies the link’s creation order position.
  LOGICAL, INTENT(OUT) :: f_corder_valid
                        ! Indicates whether the value in corder is valid.
  INTEGER, INTENT(OUT) :: link_type
                        ! Specifies the link class:
                        !  H5L_LINK_HARD_F      - Hard link
                        !  H5L_LINK_SOFT_F      - Soft link
                        !  H5L_LINK_EXTERNAL_F  - External link
                        !  H5L_LINK_ERROR _F    - Error
  INTEGER, INTENT(OUT) :: address
                        ! If the link is a hard link, address specifies the file
                        ! address that the link points to
  INTEGER(HSIZE_T), INTENT(OUT) :: val_size
                        ! If the link is a symbolic link, val_size will be the
                        ! length of the link value
  INTEGER, INTENT(OUT) :: hdferr
                        ! Error code:
                        ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                        ! Link access property list
END SUBROUTINE h5lget_info_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lget_info_by_idx

Signature:

herr_t `H5Lget_info_by_idx`(*hid_t* `loc_id`, *const char* *`group_name`, *H5_index_t* `index_field`, *H5_iter_order_t* `order`, *hsize_t* `n`, *H5L_info_t* *`link_val`, *hid_t* `lapl_id`)

Purpose:

Retrieves metadata for a link in a group, according to the order within a field or index.

Description:

`H5Lget_info_by_idx` returns the metadata for a link in a group according to a specified field or index and a specified order.

The link for which information is to be returned is specified by `index_field`, `order`, and `n` as follows:

◊ `index_field` specifies the field by which the links in `group_name` are ordered. The links may be indexed on this field, in which case operations seeking specific links are likely to complete more quickly.
◊ `order` specifies the order in which the links are to be referenced for the purposes of this function.
◊ `n` specifies the position of the subject link. Note that this count is zero-based; `0` (zero) indicates that the function will return the value of the first link; if `n` is `5`, the function will return the value of the sixth link; etc.

For example, assume that `index_field`, `order`, and `n` are `H5_INDEX_NAME`, `H5_ITER_DEC`, and `5`, respectively. `H5_INDEX_NAME` indicates that the links are accessed in alpha-numeric order by their names. `H5_ITER_DEC` specifies that the list be traversed in reverse order, or in decremented order. And `5` specifies that this call to the function will return the metadata for the 6th link (`n` + 1) from the end.

See `H5Literate` for a list of valid values and further discussion regarding `index_field` and `order`.

If `loc_id` specifies the group in which the link resides, `group_name` can be a dot (.).

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of subject group
const char *`group_name`	IN: Name of subject group
H5_index_t `index_field`	IN: Index or field which determines the order
H5_iter_order_t `order`	IN: Order within field or index
hsize_t `n`	IN: Link for which to retrieve information
H5L_info_t *`link_val`	OUT: Buffer in which link value is returned
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5lget_info_by_idx_f

```
SUBROUTINE h5lget_info_by_idx_f(loc_id, group_name, index_field, order, n, &
               f_corder_valid, corder, cset, data_size, hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                             ! File or group identifier specifying
                             ! location of subject group
  CHARACTER(LEN=*), INTENT(IN) :: group_name
                             ! Name of subject group
  INTEGER, INTENT(IN) :: index_field
                             ! Index/field which determines the order
                             !   H5_INDEX_UNKNOWN_F   - Unknown index type
                             !   H5_INDEX_NAME_F      - Index on names
                             !   H5_INDEX_CRT_ORDER_F - Index on creation order
                             !   H5_INDEX_N_F         - Number of indices defined
  INTEGER, INTENT(IN) :: order
                             ! Order in which to iterate over index;
                             ! Possible values are:
                             !    H5_ITER_UNKNOWN_F  - Unknown order
                             !    H5_ITER_INC_F      - Increasing order
                             !    H5_ITER_DEC_F      - Decreasing order
                             !    H5_ITER_NATIVE_F   - No particular order,
                             !                         whatever is fastest
  INTEGER(HSIZE_T), INTENT(IN) :: n
                             ! Attribute’s position in index
  LOGICAL, INTENT(OUT) :: f_corder_valid
                             ! Indicates whether the creation order data is
                             ! valid for this attribute
  INTEGER, INTENT(OUT) :: corder
                             ! Is a positive integer containing the creation
                             ! order of the attribute
  INTEGER, INTENT(OUT) :: cset
                             ! Indicates the character set used for the
                             ! attribute’s name
  INTEGER(HSIZE_T), INTENT(OUT) :: data_size
                             ! Indicates the size, in the number of characters,
                             ! of the attribute
  INTEGER, INTENT(OUT) :: hdferr
                             ! Error code:
                             ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                             ! Link access property list
END SUBROUTINE h5lget_info_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lget_name_by_idx

Signature:

ssize_t `H5Lget_name_by_idx`(*hid_t* `loc_id`, *const char* *`group_name`, *H5_index_t* `index_field`, *H5_iter_order_t* `order`, *hsize_t* `n`, *char* *`name`, *size_t* `size`, *hid_t* `lapl_id`)

Purpose:

Retrieves name of the *n*th link in a group, according to the order within a specified field or index.

Description:

`H5Lget_name_by_idx` retrieves the name of the *n*th link in a group, according to the specified order, `order`, within a specified field or index, `index_field`.

If `loc_id` specifies the group in which the link resides, `group_name` can be a dot (.).

The size in bytes of `name` is specified in `size`. If `size` is unknown, it can be determined via an initial `H5Lget_name_by_idx` call with `name` set to `NULL`; the function's return value will be the size of the name.

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of subject group
const char *`group_name`	IN: Name of subject group
H5_index_t `index_field`	IN: Index or field which determines the order
H5_iter_order_t `order`	IN: Order within field or index
hsize_t `n`	IN: Link for which to retrieve information
char *`name`	OUT: Buffer in which link value is returned
size_t `size`	IN: Size in bytes of `name`
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns the size of the link name if successful; otherwise returns a negative value.

Fortran90 Interface: h5lget_name_by_idx_f

```
SUBROUTINE h5lget_name_by_idx_f(loc_id, group_name, index_field, order, n, &
      name, hdferr, lapl_id, size)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                            ! File or group identifier specifying location of
                            ! subject group
  CHARACTER(LEN=*), INTENT(IN) :: group_name
                            ! Name of subject group
  INTEGER, INTENT(IN) :: index_field
                            ! Index or field which determines the order
                            !  H5_INDEX_UNKNOWN_F   - Unknown index type
                            !  H5_INDEX_NAME_F      - Index on names
                            !  H5_INDEX_CRT_ORDER_F - Index on creation order
                            !  H5_INDEX_N_F         - Number of indices defined
  INTEGER, INTENT(IN) :: order
                            ! Order in which to iterate over index:
                            !    H5_ITER_UNKNOWN_F  - Unknown order
                            !    H5_ITER_INC_F      - Increasing order
                            !    H5_ITER_DEC_F      - Decreasing order
                            !    H5_ITER_NATIVE_F   - No particular order,
                            !                         whatever is fastest
```

```
    INTEGER(HSIZE_T), INTENT(IN) :: n
                               ! Attribute’s position in index
    CHARACTER(LEN=*), INTENT(OUT) :: name
                               ! Buffer in which link value is returned
    INTEGER, INTENT(OUT) :: hdferr        ! Error code:
                               ! 0 on success and -1 on failure
    INTEGER(SIZE_T), OPTIONAL, INTENT(OUT) :: size
                               ! Indicates the size, in the number of characters,
                               ! of the link, returns exact size
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                               ! Link access property list
  END SUBROUTINE h5lget_name_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lget_val

Signature:

herr_t `H5Lget_val`(*hid_t* `link_loc_id`, *const char* `*link_name`, *void* `*linkval_buff`, *size_t* `size`, *hid_t* `lapl_id`)

Purpose:

Returns the value of a symbolic link.

Description:

`H5Lget_val` returns the link value of the link `link_name`.

The parameter `link_loc_id` is a file or group identifier.

`link_name` identifies a symbolic link and is defined relative to `link_loc_id`. Symbolic links include soft and external links and some user-defined links.

The link value is returned in the buffer `linkval_buff`. For soft links, this is the path to which the link points, including the null terminator; for external and user-defined links, it is the link buffer.

`size` is the size of `linkval_buff` and should be the size of the link value being returned. This size value can be determined through a call to `H5Lget_info`; it is returned in the `val_size` field of the `H5L_info_t` struct.

If `size` is smaller than the size of the returned value, then the string stored in `linkval_buff` will be truncated to `size` bytes. For soft links, this means that the value will not be null terminated.

In the case of external links, the target file and object names are extracted from `linkval_buff` by calling `H5Lunpack_elink_val`.

The link class of `link_name` can be determined with a call to `H5Lget_info`.

`lapl_id` specifies the link access property list associated with the link `link_name`. In the general case, when default link access properties are acceptable, this can be passed in as `H5P_DEFAULT`. An example of a situation that requires a non-default link access property list is when the link is an external link; an external link may require that a link prefix be set in a link access property list (see `H5Pset_elink_prefix`).

This function should be used only after `H5Lget_info` has been called to verify that `link_name` is a symbolic link. This can be deteremined from the `link_type` field of the `H5L_info_t` struct.

Parameters:

hid_t `link_loc_id`	IN: File or group identifier.
const char `*link_name`	IN: Link whose value is to be returned.
void `*linkval_buff`	OUT: The buffer to hold the returned link value.
size_t `size`	IN: Maximum number of characters of link value to be returned.
hid_t `lapl_id`	IN: List access property list identifier.

Returns:

Returns a non-negative value, with the link value in `linkval_buff`, if successful. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lget_val_by_idx

Signature:

herr_t `H5Lget_val_by_idx`(*hid_t* `loc_id`, *const char* *`group_name`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *hsize_t* `n`, *void* *`link_val`, *size_t* `size`, *hid_t* `lapl_id`)

Purpose:

Retrieves value of the *n*th link in a group, according to the order within an index.

Description:

`H5Lget_val_by_idx` retrieves the value of the *n*th link in a group, according to the specified order, `order`, within an index, `index`.

◊ For hard links, the value is the address of the object pointed to.
◊ For soft links, the value is the path name of the object pointed to.
◊ For external links, this is a compound value containing file and path name information; to use this external link information, it must first be decoded with `H5Lunpack_elink_val`
◊ For user-defined links, this value will be described in the definition of the user-defined link type.

If `loc_id` specifies the group in which the link resides, `group_name` can be a dot (.).

The size in bytes of `group_name` is specified in `size`. If `size` is unknown, it can be determined via an initial `H5Lget_val_by_idx` call with `size` set to `NULL`; `size` will be returned with the actual size of `group_name`.

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of subject group
const char *`group_name`	IN: Name of subject group
H5_index_t `index_type`	IN: Type of index; valid values include: `NAME` Indexed by name `CORDER` Indexed by creation order
H5_iter_order_t `order`	IN: Order within field or index; valid values include: `H5_ITER_INC` Iterate in increasing order `H5_ITER_DEC` Iterate in decreasing order `H5_ITER_NATIVE` Iterate in fastest order
hsize_t `n`	IN: Link for which to retrieve information
void *`link_val`	OUT: Pointer to buffer in which link value is returned
size_t `size`	IN: Size in bytes of `group_name`
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lis_registered

Signature:

htri_t `H5Lis_registered`(*H5L_type_t* `link_cls_id`)

Purpose:

Determines whether a class of user-defined links is registered.

Description:

`H5Lis_registered` tests whether a user-defined link class is currently registered, either by the HDF5 Library or by the user through the use of `H5Lregister`.

A link class must be registered to create new links of that type or to traverse exisitng links of that type.

Parameters:

H5L_type_t `link_cls_id`	IN: User-defined link class identifier

Returns:

Returns a positive value if the link class has been registered and zero if it is unregistered. Otherwise returns a negative value; this may mean that the identifier is not a valid user-defined class identifier.

Fortran90 Interface: H5Lis_registered_f

```
SUBROUTINE H5Lis_registered_f(link_cls_id, registered, hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: link_cls_id  ! User-defined link class identifier
  LOGICAL, INTENT(OUT) :: registered  ! .TRUE.  - if the link class is registered
                                      ! .FALSE. - if it is unregistered
  INTEGER, INTENT(OUT) :: hdferr      ! Error code:
                                      ! 0 on success and -1 on failure
END SUBROUTINE H5Lis_registered_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Literate

Signature:

herr_t `H5Literate`(*hid_t* `group_id`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *hsize_t* *`idx`, *H5L_iterate_t* `op`, *void* *`op_data`)

Purpose:

Iterates through links in a group.

Description:

`H5Literate` iterates through the links in a group, specified by `group_id`, in the order of the specified index, `index_type`, using a user-defined callback routine `op`. `H5Literate` does not recursively follow links into subgroups of the specified group.

Three parameters are used to manage progress of the iteration: `index_type`, `order`, and `idx`.

`index_type` specifies the index to be used. If the links have not been indexed by the index type, they will first be sorted by that index then the iteration will begin; if the links have been so indexed, the sorting step will be unnecesary, so the iteration may begin more quickly. Valid values include the following:

`H5_INDEX_NAME`	Alpha-numeric index on name
`H5_INDEX_CRT_ORDER`	Index on creation order

`order` specifies the order in which objects are to be inspected along the index specified in `index_type`. Valid values include the following:

`H5_ITER_INC`	Increasing order
`H5_ITER_DEC`	Decreasing order
`H5_ITER_NATIVE`	Fastest available order

`idx` allows an interrupted iteration to be resumed; it is passed in by the application with a starting point and returned by the library with the point at which the iteration stopped.

The `op` callback funtion, the related *H5L_info_t* struct, and the effect of the callback function's return value on the application are described in `H5Lvisit`.

`op_data` is a user-defined pointer to the data required to process links in the course of the iteration. This pointer is passed back to each step of the iteration in the `op` callback function's `op_data` parameter.

As mentioned above, `H5Literate` is not recursive. In particular, if a member of `group_id` is found to be a group, call it `subgroup_a`, `H5Literate` does not examine the members of `subgroup_a`. When recursive iteration is required, the application can do either of the following:

◊ Use one of the following recursive routines instead of `H5Literate`:
  `H5Lvisit`
  `H5Lvisit_by_name`
  `H5Ovisit`
  `H5Ovisit_by_name`

◊ Handle the recursion manually, explicitly calling `H5Literate` on discovered subgroups.

`H5Literate` is the same as `H5Giterate`, except that `H5Giterate` always proceeds in alphanumeric order.

Parameters:

hid_t `group_id`	IN: Identifier specifying subject group
H5_index_t `index_type`	IN: Type of index which determines the order
H5_iter_order_t `order`	IN: Order within index
hsize_t `*idx`	IN: Iteration position at which to start OUT: Position at which an interrupted iteration may be restarted
H5L_iterate_t `op`	IN: Callback function passing data regarding the link to the calling application
void `*op_data`	IN: User-defined pointer to data required by the application for its processing of the link

Returns:

On success, returns the return value of the first operator that returns a positive value, or zero if all members were processed with no operator returning non-zero.

On failure, returns a negative value if something goes wrong within the library, or the first negative value returned by an operator.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Literate_by_name

Signature:

herr_t `H5Literate_by_name(` *hid_t* `loc_id`, *const char* `*group_name`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *hsize_t* `*idx`, *H5L_iterate_t* `op`, *void* `*op_data`, *hid_t* `*lapl_id`)

Purpose:

Iterates through links in a group.

Description:

`H5Literate_by_name` iterates through the links in a group, specified by `loc_id` and `group_name`, in the order of the specified index, `index_type`, using a user-defined callback routine `op`. `H5Literate_by_name` does not recursively follow links into subgroups of the specified group.

`index_type` specifies the index to be used. If the links have not been indexed by the index type, they will first be sorted by that index then the iteration will begin; if the links have been so indexed, the sorting step will be unnecesary, so the iteration may begin more quickly. Valid values include the following:

`H5_INDEX_NAME`	Alpha-numeric index on name
`H5_INDEX_CRT_ORDER`	Index on creation order

`order` specifies the order in which objects are to be inspected along the index specified in `index_type`. Valid values include the following:

`H5_ITER_INC`	Increasing order
`H5_ITER_DEC`	Decreasing order
`H5_ITER_NATIVE`	Fastest available order

`idx` allows an interrupted iteration to be resumed; it is passed in by the application with a starting point and returned by the library with the point at which the iteration stopped.

`H5Literate_by_name` is not recursive. In particular, if a member of `group_name` is found to be a group, call it `subgroup_a`, `H5Literate_by_name` does not examine the members of `subgroup_a`. When recursive iteration is required, the application must handle the recursion, explicitly calling `H5Literate_by_name` on discovered subgroups.

`H5Literate_by_name` is the same as `H5Giterate`, except that `H5Giterate` always proceeds in alphanumeric order.

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of subject group
const char `*group_name`	IN: Name of subject group
H5_index_t `index_type`	IN: Type of index which determines the order
H5_iter_order_t `order`	IN: Order within index
hsize_t `*idx`	IN: Iteration position at which to start OUT: Position at which an interrupted iteration may be restarted
H5L_iterate_t `op`	IN: Callback function passing data regarding the link to the calling application
void `*op_data`	IN: User-defined pointer to data required by the application for its processing of the link
hid_t `lapl_id`	IN: Link access property list

Returns:

On success, returns the return value of the first operator that returns a positive value, or zero if all members were processed with no operator returning non-zero.

On failure, returns a negative value if something goes wrong within the library, or the first negative value returned by an operator.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lmove

Signature:

herr_t `H5Lmove`(*hid_t* `src_loc_id`, *const char* `*src_name`, *hid_t* `dest_loc_id`, *const char* `*dest_name`, *hid_t* `lcpl`, *hid_t* `lapl`)

Purpose:

Renames a link within an HDF5 file.

Description:

`H5Lmove` renames a link within an HDF5 file. The original link, `src_name`, is removed from the group graph and the new link, `dest_name`, is inserted; this change is accomplished as an atomic operation.

`src_loc_id` and `src_name` identify the existing link. `src_loc_id` is either a file or group identifier; `src_name` is the path to the link and is interpreted relative to `src_loc_id`.

`dest_loc_id` and `dest_name` identify the new link. `dest_loc_id` is either a file or group identifier; `dest_name` is the path to the link and is interpreted relative to `dest_loc_id`.

`lcpl` and `lapl` are the link creation and link access property lists, respectively, associated with the new link, `dest_name`.

Through these property lists, several properties are available to govern the behavior of `H5Lmove`. The property controlling creation of missing intermediate groups is set in the link creation property list with `H5Pset_create_intermediate_group`; `H5Lmove` ignores any other properties in the link creation property list. Properties controlling character encoding, link traversals, and external link prefixes are set in the link access property list with `H5Pset_char_encoding`, `H5Pset_nlinks`, and `H5Pset_elink_prefix`, respectively.

Warning:

Exercise care in moving links as it is possible to render data in a file inaccessible with `H5Lmove`. If the link being moved is on the only path leading to an HDF5 object, that object may become permanently inaccessible in the file.

Parameters:

hid_t `src_loc_id`	IN: Original file or group identifier.
const char `*src_name`	IN: Original link name.
hid_t `dest_loc_id`	IN: Destination file or group identifier.
const char `*dest_name`	IN: New link name.
hid_t `lcpl_id`	IN: Link creation property list identifier to be associated with the new link.
hid_t `lapl_id`	IN: Link access property list identifier to be associated with the new link.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5lmove_f

```
SUBROUTINE h5lmove_f(src_loc_id, src_name, dest_loc_id, dest_name, hdferr, &
    lcpl_id, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: src_loc_id
                                         ! Original file or group identifier.
  CHARACTER(LEN=*), INTENT(IN) :: src_name
                                         ! Original link name.
  INTEGER(HID_T), INTENT(IN) :: dest_loc_id
                                         ! Destination file or group identifier.
    CHARACTER(LEN=*), INTENT(IN) :: dest_name
                                        ! new link name.
    INTEGER(HID_T), INTENT(OUT) :: hdferr ! Error code:
                                        ! 0 on success and -1 on failure
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                                        ! Link creation property list identifier
                                        ! to be associated with the new link.
    INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                        ! Link access property list identifier
                                        ! to be associated with the new link.
  END SUBROUTINE h5lmove_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lregister

Signature:

herr_t `H5Lregister`(*const H5L_class_t* * `link_class`)

Purpose:

Registers user-defined link class or changes behavior of existing class.

Description:

`H5Lregister` registers a class of user-defined links, or changes the behavior of an existing class.

The struct `H5L_class_t` is defined in `H5Lpublic.h` as follows:

```
  typedef struct H5L_class_t {
      int version;                    /* Version number of this struct  */
      H5L_type_t class_id;            /* Link class identifier          */
      const char *comment;            /* Comment for debugging          */
      H5L_create_func_t create_func;  /* Callback during link creation  */
      H5L_move_func_t move_func;      /* Callback after moving link     */
      H5L_copy_func_t copy_func;      /* Callback after copying link    */
      H5L_traverse_func_t trav_func;  /* The main traversal function    */
      H5L_delete_func_t del_func;     /* Callback for link deletion     */
      H5L_query_func_t query_func;    /* Callback for queries           */
  } H5L_class_t;
```

The link class passed in will override any existing link class for the specified link class identifier `class_id`. The class definition must include at least a `H5L_class_t` version (which should be `H5L_LINK_CLASS_T_VERS`), a link class identifier, and a traversal function, `trav_func`.

Valid values of `class_id` include the following (defined in `H5Lpublic.h`:

`H5L_TYPE_HARD`	Hard link
`H5L_TYPE_SOFT`	Soft link
`H5L_TYPE_EXTERNAL`	External link

`class_id` must be a value between `H5L_LINK_UD_MIN` and `H5L_LINK_UD_MAX`, Note that as distributed with the HDF5 Library, the external link class is implemented as an example user-defined link class and `H5L_LINK_EXTERNAL` equals `H5L_LINK_UD_MIN`. Therefore, `class_id` should not equal `H5L_LINK_UD_MIN` unless you intend to overwrite or modify the behavior of external links. To summarize:

`H5L_TYPE_ERROR` indicates that an error has occurred.

`H5L_TYPE_MAX` is the maximum allowed value for a link type identifier.

`H5L_TYPE_UD_MIN` equals `H5L_TYPE_EXTERNAL`.

`H5L_TYPE_UD_MAX` equals `H5L_TYPE_MAX`.

`H5L_TYPE_HARD` and `H5L_TYPE_SOFT` reside in the reserved space below `H5L_TYPE_UD_MIN`.

Note:

If you plan to distribute files with a new user-defined link class, please contact the Help Desk at The HDF Group to help prevent collisions between `class_id` values.

Parameters:

const H5L_class_t * `link_class`	IN: Struct describing user-defined link class

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ldelete

Signature:

herr_t `H5Ldelete`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `lapl_id`)

Purpose:

Removes a link from a group.

Description:

`H5Ldelete` removes the link specified by `name` from the location `loc_id`.

If the link being removed is a hard ink, `H5Ldelete` also decrements the link count for the object to which `name` points. Unless there is a duplicate hard link in that group, this action removes the object to which `name` points from the group that previously contained it.

Object headers keep track of how many hard links refer to an object; when the hard link count, also referred to as the reference count, reaches zero, the object can be removed from the file. The file space associated will then be released, i.e., identified in memory as freespace. Objects which are open are not removed until all identifiers to the object are closed.

Note that space identified as freespace is available for re-use only as long as the file remains open; once a file has been closed, the HDF5 library loses track of freespace. See “Freespace Management” in “Performace Analysis and Issues” for further details.

Warning:

Exercise caution in the use of `H5Ldelete`; if the link being removed is on the only path leading to an HDF5 object, that object may become permanently inaccessible in the file.

Parameters:

hid_t `loc_id`	IN: Identifier of the file or group containing the object.
const char *`name`	IN: Name of the link to delete.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5ldelete_f

```
SUBROUTINE h5ldelete_f(loc_id, name, hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id  ! Identifier of the file or group
                                        ! containing the object
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of the link to delete
  INTEGER, INTENT(OUT) :: hdferr        ! Error code:
                                        ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                        ! Link access property list identifier
END SUBROUTINE h5ldelete_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ldelete_by_idx

Signature:

herr_t `H5Ldelete_by_idx`(*hid_t* `loc_id`, *const char* *`group_name`, *H5_index_t* `index_field`, *H5_iter_order_t* `order`, *hsize_t* `n`, *hid_t* `lapl_id`)

Purpose:

Removes the *n*th link in a group.

Description:

`H5Ldelete_by_idx` removes the *n*th link in a group according to the specified order, `order`, in the specified index, `index`.

If `loc_id` specifies the group in which the link resides, `group_name` can be a dot (.).

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of subject group
const char *`group_name`	IN: Name of subject group
H5_index_t `index_field`	IN: Index or field which determines the order
H5_iter_order_t `order`	IN: Order within field or index
hsize_t `n`	IN: Link for which to retrieve information
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5ldelete_by_idx_f

```
SUBROUTINE h5ldelete_by_idx_f(loc_id, group_name, index_field, order, n, &
     hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                         ! Identifer for object to which attribute is attached.
  CHARACTER(LEN=*), INTENT(IN) :: group_name
                         ! Name of object, relative to location,
                         ! from which attribute is to be removed
  INTEGER, INTENT(IN) :: index_field
                         ! Type of index; Possible values are:
                         !    H5_INDEX_UNKNOWN_F   - Unknown index type
                         !    H5_INDEX_NAME_F      - Index on names
                         !    H5_INDEX_CRT_ORDER_F - Index on creation order
                         !    H5_INDEX_N_F          - Number of indices defined
  INTEGER, INTENT(IN) :: order
                         ! Order in which to iterate over index;
                         ! Possible values are:
                         !    H5_ITER_UNKNOWN_F  - Unknown order
                         !    H5_ITER_INC_F      - Increasing order
                         !    H5_ITER_DEC_F      - Decreasing order
                         !    H5_ITER_NATIVE_F   - No particular order,
                         !                         whatever is fastest
                         !    H5_ITER_N_F        - Number of iteration orders
  INTEGER(HSIZE_T), INTENT(IN) :: n
                         ! Offset within index
  INTEGER, INTENT(OUT) :: hdferr
                         ! Error code:
                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                         ! Link access property list
END SUBROUTINE h5ldelete_by_idx_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lunpack_elink_val

Signature:

herr_t `H5Lunpack_elink_val`(*char* *`ext_linkval`, *size_t* `link_size`, *unsigned* *`flags`, *const char* **`filename`, *const char* **`obj_path`)

Purpose:

Decodes external link information.

Description:

`H5Lunpack_elink_val` decodes the external link information returned by `H5Lget_val` in the `ext_linkval` buffer.

`ext_linkval` should be the buffer set by `H5Lget_val` and will consist of two NULL-terminated strings, the filename and object path, one after the other.

Given this buffer, `H5Lunpack_elink_val` creates pointers to the filename and object path within the buffer and returns them in `filename` and `obj_path`, unless they are passed in as NULL.

`H5Lunpack_elink_val` requires that `ext_linkval` contain a concatenated pair of null-terminated strings, so use of this function on a string that is not an external link `udata` buffer may result in a segmentation fault. This failure can be avoided by adhering to the following procedure:

1. Call `H5Lget_info` to get the link type and the size of the link value.
2. Verify that the link is an external link, i.e., that its link type is `H5L_LINK_EXTERNAL`.
3. Call `H5Lget_val` to get the link value.
4. Call `H5Lunpack_elink_val` to unpack that value.

Parameters:

const char *`ext_linkval`	IN: Buffer containing external link information
size_t `link_size`	IN: Size, in bytes, of the `ext_linkval` buffer
unsigned *`flags`	OUT: External link flags, packed as a bitmap *(Reserved as a bitmap for flags; no flags are currently defined, so the only valid value is 0.)*
const char **`filename`	OUT: Returned filename
const char **`obj_path`	OUT: Returned object path, relative to `filename`

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lunregister

Signature:

herr_t `H5Lunregister(` *H5L_type_t* `link_cls_id`)

Purpose:

Unregisters a class of user-defined links.

Description:

`H5Lunregister` unregisters a class of user-defined links, preventing them from being traversed, queried, moved, etc.

A link class can be re-registered using `H5Lregister`.

Parameters:

H5L_type_t `link_cls_id`	IN: User-defined link class identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lvisit
Signature:

herr_t `H5Lvisit(` *hid_t* `group_id`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *H5L_iterate_t* `op`, *void* `*op_data`)

Purpose:

Recursively visits all links starting from a specified group.

Description:

`H5Lvisit` is a recursive iteration function to visit all links in and below a group in an HDF5 file, thus providing a mechanism for an application to perform a common set of operations across all of those links or a dynamically selected subset. For non-recursive iteration across the members of a group, see `H5Literate`.

The group serving as the root of the iteration is specified by its group identifier, `group_id`

Two parameters are used to establish the iteration: `index_type` and `order`.

`index_type` specifies the index to be used. If the links have not been indexed by the index type, they will first be sorted by that index then the iteration will begin; if the links have been so indexed, the sorting step will be unnecesary, so the iteration may begin more quickly. Valid values include the following:

`H5_INDEX_NAME`	Alpha-numeric index on name
`H5_INDEX_CRT_ORDER`	Index on creation order

Note that the index type passed in `index_type` is a *best effort* setting. If the application passes in a value indicating iteration in creation order and a group is encountered that was not tracked in creation order, that group will be iterated over in alpha-numeric order by name, or *name order*. (*Name order* is the native order used by the HDF5 Library and is always available.)

`order` specifies the order in which objects are to be inspected along the index specified in `index_type`. Valid values include the following:

`H5_ITER_INC`	Increasing order
`H5_ITER_DEC`	Decreasing order
`H5_ITER_NATIVE`	Fastest available order

The protoype of the callback function `op` is as follows (as defined in the source code file `H5Lpublic.h`):

herr_t `(*H5L_iterate_t)(` *hid_t* `g_id,` *const char* `*name,` *const H5L_info_t* `*info,` *void* `*op_data)`

The parameters of this callback function have the following values or meanings:

`g_id`	Group that serves as root of the iteration; same value as the `H5Lvisit` `group_id` parameter
`name`	Name of link, relative to `g_id`, being examined at current step of the iteration
`info`	*H5L_info_t* struct containing information regarding that link

`op_data`	User-defined pointer to data required by the application in processing the link; a pass-through of the `op_data` pointer provided with the `H5Lvisit` function call

The *H5L_info_t* struct is defined (in `H5Lpublic.h`) as follows:

```
typedef struct {
    H5L_type_t     type;         /* Type of link                   */
    hbool_t        corder_valid; /* Indicates whether creation     */
                                 /* order is valid                 */
    int64_t        corder;       /* Creation order                 */
    H5T_cset_t     cset;         /* Character set of link name     */
    union {
        haddr_t    address;      /* Address hard link points to    */
        size_t     val_size;     /* Size of soft link or           */
                                 /* user-defined link value        */
    } u;
} H5L_info_t;
```

The possible return values from the callback function, and the effect of each, are as follows:

◊ Zero causes the visit iterator to continue, returning zero when all group members have been processed.
◊ A positive value causes the visit iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the next group member.
◊ A negative value causes the visit iterator to immediately return that value, indicating failure. The iterator can be restarted at the next group member.

The `H5Lvisit` `op_data` parameter is a user-defined pointer to the data required to process links in the course of the iteration. This pointer is passed back to each step of the iteration in the `op` callback function's `op_data` parameter.

`H5Lvisit` and `H5Ovisit` are companion functions: one for examining and operating on links; the other for examining and operating on the objects that those links point to. Both functions ensure that by the time the function completes successfully, every link or object below the specified point in the file has been presented to the application for whatever processing the application requires.

Parameters:

hid_t `group_id`	IN: Identifier of the group at which the recursive iteration begins.

H5_index_t `index_type`	IN: Type of index; valid values include:
	`H5_INDEX_NAME`
	`H5_INDEX_CRT_ORDER`

H5_iter_order_t `order`	IN: Order in which index is traversed; valid values include:
	`H5_ITER_DEC`
	`H5_ITER_INC`
	`H5_ITER_NATIVE`

H5L_iterate_t `op`	IN: Callback function passing data regarding the link to the calling application
void *`op_data`	IN: User-defined pointer to data required by the application for its processing of the link

Returns:

On success, returns the return value of the first operator that returns a positive value, or zero if all members were processed with no operator returning non-zero.

On failure, returns a negative value if something goes wrong within the library, or the first negative value returned by an operator.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Lvisit_by_name

Signature:

herr_t `H5Lvisit_by_name(` *hid_t* `loc_id`, *const char* `*group_name`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *H5L_iterate_t* `op`, *void* `*op_data`, *hid_t* `lapl_id`)

Purpose:

Recursively visits all links starting from a specified group.

Description:

`H5Lvisit_by_name` is a recursive iteration function to visit all links in and below a group in an HDF5 file, thus providing a mechanism for an application to perform a common set of operations across all of those links or a dynamically selected subset. For non-recursive iteration across the members of a group, see `H5Literate`.

The group serving as the root of the iteration is specified by the `loc_id` / `group_name` parameter pair. `loc_id` specifies a file or group; `group_name` specifies either a group in the file (with an absolute name based in the file's root group) or a group relative to `loc_id`. If `loc_id` fully specifies the group that is to serve as the root of the iteration, `group_name` should be `'.'` (a dot). (Note that when `loc_id` fully specifies the the group that is to serve as the root of the iteration, the user may wish to consider using `H5Lvisit` instead of `H5Lvisit_by_name`.)

Two parameters are used to establish the iteration: `index_type` and `order`.

`index_type` specifies the index to be used. If the links have not been indexed by the index type, they will first be sorted by that index then the iteration will begin; if the links have been so indexed, the sorting step will be unnecesary, so the iteration may begin more quickly. Valid values include the following:

`H5_INDEX_NAME`	Alpha-numeric index on name
`H5_INDEX_CRT_ORDER`	Index on creation order

Note that the index type passed in `index_type` is a *best effort* setting. If the application passes in a value indicating iteration in creation order and a group is encountered that was not tracked in creation order, that group will be iterated over in alpha-numeric order by name, or *name order*. (*Name order* is the native order used by the HDF5 Library and is always available.)

`order` specifies the order in which objects are to be inspected along the index specified in `index_type`. Valid values include the following:

`H5_ITER_INC`	Increasing order
`H5_ITER_DEC`	Decreasing order
`H5_ITER_NATIVE`	Fastest available order

The `op` callback funtion, the related *H5L_info_t* struct, and the effect that the callback function's return value has on the application are described in `H5Lvisit`.

The `H5Lvisit_by_name` `op_data` parameter is a user-defined pointer to the data required to process links in the course of the iteration. This pointer is passed back to each step of the iteration in the callback function's `op_data` parameter.

`lapl_id` is a link access property list. In the general case, when default link access properties are acceptable, this can be passed in as `H5P_DEFAULT`. An example of a situation that requires a non-default link access property list is when the link is an external link; an external link may require that a link prefix be set in a link access property list (see `H5Pset_elink_prefix`).

`H5Lvisit_by_name` and `H5Ovisit_by_name` are companion functions: one for examining and operating on links; the other for examining and operating on the objects that those links point to. Both functions ensure that by the time the function completes successfully, every link or object below the specified point in the file has been presented to the application for whatever processing the application requires.

Parameters:

hid_t `loc_id`	IN: Identifier of a file or group
const char `*name`	IN: Name of the group, generally relative to `loc_id`, that will serve as root of the iteration

H5_index_t `index_type`	IN: Type of index; valid values include:
	`H5_INDEX_NAME`
	`H5_INDEX_CRT_ORDER`

H5_iter_order_t `order`	IN: Order in which index is traversed; valid values include:
	`H5_ITER_DEC`
	`H5_ITER_INC`
	`H5_ITER_NATIVE`

H5L_iterate_t `op`	IN: Callback function passing data regarding the link to the calling application
void `*op_data`	IN: User-defined pointer to data required by the application for its processing of the link
hid_t `lapl_id`	IN: Link access property list identifier

Returns:

On success, returns the return value of the first operator that returns a positive value, or zero if all members were processed with no operator returning non-zero.

On failure, returns a negative value if something goes wrong within the library, or the first negative value returned by an operator.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

H5O: Object Interface

Object API Functions

The Object interface, H5O, functions manipulate objects in an HDF5 file. This interface is designed to be used in conjunction with the Links interface (H5L).

The C Interfaces:

- H5Oopen
- H5Olink
- H5Oclose
- H5Ocopy
- H5Ovisit
- H5Ovisit_by_name
- H5Oset_comment
- H5Oset_comment_by_name
- H5Oget_comment
- H5Oget_comment_by_name
- H5Oget_info
- H5Oget_info_by_name
- H5Oget_info_by_idx
- H5Oopen_by_idx
- H5Oopen_by_addr
- H5Oincr_refcount
- H5Odecr_refcount

Alphabetical Listing

- H5Oclose
- H5Ocopy
- H5Odecr_refcount
- H5Oget_comment
- H5Oget_comment_by_name
- H5Oget_info
- H5Oget_info_by_idx
- H5Oget_info_by_name
- H5Oincr_refcount
- H5Olink
- H5Oopen
- H5Oopen_by_addr
- H5Oopen_by_idx
- H5Oset_comment
- H5Oset_comment_by_name
- H5Ovisit
- H5Ovisit_by_name

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5olink_f
- h5oopen_f

Name: H5Oclose

Signature:

herr_t `H5Oclose`(*hid_t* `object_id`)

Purpose:

Closes an object in an HDF5 file.

Description:

`H5Oclose` closes the group, dataset, or named datatype specified by `object_id`.

This function is the companion to `H5Oopen`, and has the same effect as calling `H5Gclose`, `H5Dclose`, or `H5Tclose`.

`H5Oclose` is *not* used to close a dataspace, attribute, property list, or file.

Parameters:

hid_t `object_id`	IN: Object identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ocopy

Signature:

herr_t `H5Ocopy`(*hid_t* `src_loc_id`, *const char* `*src_name`, *hid_t* `dst_loc_id`, *const char* `*dst_name`, *hid_t* `ocpypl_id`, *hid_t* `lcpl_id`)

Purpose:

Copies an object in an HDF5 file.

Description:

`H5Ocopy` copies the group, dataset or named datatype specified by `src_name` from the file or group specified by `src_loc_id` to the destination location `dst_loc_id`.

The destination location, as specified in `dst_loc_id`, may be a group in the current file or a location in a different file. If `dst_loc_id` is a file identifier, the copy will be placed in that file's root group.

The new copy will be created with the name `dst_name`. `dst_name` must not pre-exist in the destination location; if `dst_name` already exists at the location `dst_loc_id`, `H5Ocopy` will fail.

The new copy of the object is created with the creation property lists specified by `ocpypl_id` and `lcpl_id`.

Several flags are available to govern the behavior of `H5Ocopy`. These flags are set in the creation property list `cplist_id` with `H5Pset_copy_object` and `H5Pset_create_intermediate_group`. All of the available flags are described at `H5Pset_copy_object`.

Parameters:

hid_t `src_loc_id`	Object identifier indicating the location of the source object to be copied
const char `*src_name`	Name of the source object to be copied
hid_t `dst_loc_id`	Location identifier specifying the destination
const char `*dst_name`	Name to be assigned to the new copy
hid_t `ocpypl_id`	Object copy property list
hid_t `lcpl_id`	Link creation property list for the new hard link

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Odecr_refcount

Signature:

herr_t `H5Odecr_refcount(` *hid_t* `object_id`)

Purpose:

Decrements an object reference count.

Description:

`H5Odecr_refcount` decrements the hard link reference count for an object. It should be used any time a user-defined link that references an object by address is deleted. In general, `H5Oincr_refcount` will have been used previously, when the link was created.

An object's *reference count* is the number of hard links in the file that point to that object. See the "Programming Model" section of the "HDF5 Groups" chapter in the *HDF5 User's Guide* for a more complete discussion of reference counts.

If a user application needs to determine an object's reference count, an `H5Oget_info` call is required; the reference count is returned in the `rc` field of the `H5O_info_t` struct.

Warning: This function must be used with care!
Improper use can lead to inaccessible data, wasted space in the file, or ***file corruption***.

Parameters:

hid_t `object_id`	IN: Object identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oget_comment

Signature:

ssize_t `H5Oget_comment`(*hid_t* `object_id`, *char* *`comment`, *size_t* `bufsize`,)

Purpose:

Retrieves comment for specified object.

Description:

`H5Oget_comment` retrieves the comment for the specified object in the buffer `comment`.

The target object is specified by an identifier, `object_id`.

The size in bytes of the comment, including the `NULL` terminator, is specified in `bufsize`. If `bufsize` is unknown, a preliminary `H5Oget_comment` call with the pointer `comment` set to `NULL` will return the size of the comment *without* the `NULL` terminator.

If `bufsize` is set to a smaller value than described above, only `bufsize` bytes of the comment, without a `NULL` terminator, are returned in `comment`.

If an object does not have a comment, the empty string is returned in `comment`.

Parameters:

hid_t `object_id`	IN: Identifier for the target object.
char *`comment`	OUT: The comment.
size_t `bufsize`	IN: Anticipated required size of the `comment` buffer.

Returns:

Upon success, returns the number of characters in the comment, not including the `NULL` terminator, or zero (`0`) if the object has no comment. The value returned may be larger than `bufsize`. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oget_comment_by_name

Signature:

ssize_t `H5Oget_comment_by_name`(*hid_t* `loc_id`, *const char* *`name`, *char* *`comment`, *size_t* `bufsize`, *hid_t* `lapl_id`)

Purpose:

Retrieves comment for specified object.

Description:

`H5Oget_comment_by_name` retrieves the comment for an object in the buffer `comment`.

The target object is specified by `loc_id` and `name`. `loc_id` can specify any object in the file. `name` can be one of the following:
— The name of the object relative to `loc_id`
— An absolute name of the object, starting from `/`, the file's root group
— A dot (`.`), if `loc_id` fully specifies the object

The size in bytes of the comment, including the `NULL` terminator, is specified in `bufsize`. If `bufsize` is unknown, a preliminary `H5Oget_comment_by_name` call with the pointer `comment` set to `NULL` will return the size of the comment *without* the `NULL` terminator.

If `bufsize` is set to a smaller value than described above, only `bufsize` bytes of the comment, without a `NULL` terminator, are returned in `comment`.

If an object does not have a comment, the empty string is returned in `comment`.

`lapl_id` contains a link access property list identifier. A link access propety list can come into play when traversing links to access an object.

Parameters:

hid_t `loc_id`	IN: Identifier of a file, group, dataset, or named datatype.
const char *`name`	IN: Name of the object whose comment is to be retrieved, specified as a path relative to `loc_id`. `name` can be '.' (a dot) if `loc_id` fully specifies the object for which the associated comment is to be retrieved.
char *`comment`	OUT: The comment.
size_t `bufsize`	IN: Anticipated required size of the `comment` buffer.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Upon success, returns the number of characters in the comment, not including the `NULL` terminator, or zero (`0`) if the object has no comment. The value returned may be larger than `bufsize`. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oget_info

Signature:

herr_t `H5Oget_info`(*hid_t* `object_id`, *H5O_info_t* *`object_info`)

Purpose:

Retrieves the metadata for an object specified by an identifier.

Description:

`H5Oget_info` specifies an object by its identifier, `object_id`, and retrieves the metadata describing that object in `object_info`, an *H5O_info_t* struct.

An *H5O_info_t* struct is defined (in `H5Opublic.h`) as follows :

```
typedef struct H5O_info_t {
    unsigned long       fileno;         /* File number that object is         */
                                        /*    located in                      */
    haddr_t             addr;           /* Object address in file             */
    H5O_type_t          type;           /* Basic object type (group,          */
                                        /*    dataset, etc.)                  */
    unsigned            rc;             /* Reference count of object          */
    time_t              atime;          /* Access time                        */
    time_t              mtime;          /* Modification time                  */
    time_t              ctime;          /* Change time                        */
    time_t              btime;          /* Birth time                         */
    hsize_t             num_attrs;      /* # of attributes attached to object */
    struct {
        unsigned version;               /* Version number of header format in */
                                        /*    file                            */
        unsigned nmesgs;                /* Number of object header messages   */
        unsigned nchunks;               /* Number of object header chunks     */
        unsigned flags;                 /* Object header status flags         */
        struct {
            hsize_t total;              /* Total space for storing object     */
                                        /*    header in file                  */
            hsize_t meta;               /* Space within header for object     */
                                        /*    header metadata information     */
            hsize_t mesg;               /* Space within header for actual     */
                                        /*    message information             */
            hsize_t free;               /* Free space within object header    */
        } space;
        struct {
            uint64_t present;           /* Flags to indicate presence of      */
                                        /*    message type in header          */
            uint64_t shared;            /* Flags to indicate message type is  */
                                        /*    shared in header                */
        } mesg;
    } hdr;
    /* Extra metadata storage for obj & attributes */
    struct {
        H5_ih_info_t   obj;             /* v1/v2 B-tree & local/fractal heap  */
                                        /*    for groups, B-tree for chunked  */
                                        /*    datasets                        */
        H5_ih_info_t   attr;            /* v2 B-tree & heap for attributes    */
    } meta_size;
} H5O_info_t;
```

Parameters:

hid_t `object_id`	IN: Identifier for target object
H5O_info_t `*object_info`	OUT: Buffer in which to return object information

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oget_info_by_idx

Signature:

herr_t `H5Oget_info_by_idx`(*hid_t* `loc_id`, *const char* *`group_name`, *H5_index_t* `index_field`, *H5_iter_order_t* `order`, *hsize_t* `n`, *H5O_info_t* *`object_info`, *hid_t* `lapl_id`)

Purpose:

Retrieves the metadata for an object, identifying the object by an index position.

Description:

`H5Oget_info_by_idx` specifies a location, `loc_id`; a group name, `group_name`; an index by which obects in that group are tracked, `index_field`; the order by which the index is to be traversed, `order`; and an object's position `n` within that index and retrieves the metadata describing that object in the struct `object_info`.

`object_info`, in which the object information is returned, is a struct of type *H5O_info_t*. This struct type is described in the `H5Oget_info` function entry.

If `loc_id` fully specifies the group in which the object resides, `group_name` can be a dot (.).

The link access property list, `lapl_id`, is not currently used; it should be passed in as `NULL`.

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of group in which object is located
const char *`group_name`	IN: Name of group in which object is located
H5_index_t `index_field`	IN: Index or field that determines the order
H5_iter_order_t `order`	IN: Order within field or index
hsize_t `n`	IN: Object for which information is to be returned
H5O_info_t *`object_info`	OUT: Buffer in which to return object information
hid_t `lapl_id`	IN: Link access property list *(Not currently used; pass as `NULL`.)*

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oget_info_by_name

Signature:

herr_t `H5Oget_info_by_name`(*hid_t* `loc_id`, *const char* *`object_name`, *H5O_info_t* *`object_info`, *hid_t* `lapl_id`)

Purpose:

Retrieves the metadata for an object, identifying the object by location and relative name.

Description:

`H5Oget_info_by_name` specifies an object's location and name, `loc_id` and `object_name`, respectively, and retrieves the metadata describing that object in `object_info`, an *H5O_info_t* struct.

The struct *H5O_info_t* is defined in `H5Opublic.h` and described in the `H5Oget_info` function entry.

The link access property list, `lapl_id`, is not currently used; it should be passed in as `H5P_DEFAULT`.

Parameters:

hid_t `loc_id`	IN: File or group identifier specifying location of group in which object is located
const char *`name`	IN: Name of group, relative to `loc_id`
H5O_info_t *`object_info`	OUT: Buffer in which to return object information
hid_t `lapl_id`	IN: Link access property list *(Not currently used; pass as `H5P_DEFAULT`.)*

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oincr_refcount

Signature:

herr_t `H5Oincr_refcount(` *hid_t* `object_id`)

Purpose:

Increments an object reference count.

Description:

`H5Oincr_refcount` increments the hard link reference count for an object. It should be used any time a user-defined link that references an object by address is added. When the link is deleted, `H5Odecr_refcount` should be used.

An object's *reference count* is the number of hard links in the file that point to that object. See the "Programming Model" section of the "HDF5 Groups" chapter in the *HDF5 User's Guide* for a more complete discussion of reference counts.

If a user application needs to determine an object's reference count, an `H5Oget_info` call is required; the reference count is returned in the `rc` field of the `H5O_info_t` struct.

Warning: This function must be used with care!
Improper use can lead to inaccessible data, wasted space in the file, or ***file corruption***.

Parameters:

hid_t `object_id`	IN: Object identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Olink

Signature:

herr_t `H5Olink`(*hid_t* `object_id`, *hid_t* `new_loc_id`, *const char* `*new_link_name`, *hid_t* `lcpl`, *hid_t* `lapl`)

Purpose:

Creates a hard link to an object in an HDF5 file.

Description:

`H5Olink` creates a new hard link to an object in an HDF5 file.

`new_loc_id` and `new_name` specify the location and name of the new link while `object_id` identifies the object that the link points to.

`H5Olink` is designed for two purposes:

- To create the first hard link to an object that has just been created with one of the `H5*create_anon` functions or with `H5Tcommit_anon`.
- To add additional structure to an existing file so that, for example, an object can be shared among multiple groups.

`lcpl` and `lapl` are the link creation and access property lists associated with the new link.

Parameters:

hid_t `object_id`	IN: Object to be linked.
hid_t `new_loc_id`	IN: File or group identifier specifying location at which object is to be linked.
const char `*new_link_name`	IN: Name of link to be created, relative to `new_loc_id`.
hid_t `lcpl_id`	IN: Link creation property list identifier.
hid_t `lapl_id`	IN: Link access property list identifier.

Example:

To create a new link to an object while simultaneously creating missing intermediate groups:
Suppose that an application must create the group `C` with the path `/A/B01/C` but may not know at run time whether the groups `A` and `B01` exist. The following code ensures that those groups are created if they are missing:

```
hid_t lcpl_id = H5Pcreate(H5P_LINK_CREATE);   /* Creates a link creation
                                               * property list (LCPL).    */
int   status  = H5Pset_create_intermediate_group(lcpl_id, TRUE);
                                       /* Sets "create missing intermediate
                                        * groups" property in that LCPL.  */
hid_t gid     = H5Gcreate_anon(file_id, H5P_DEFAULT, H5P_DEFAULT);
                                       /* Creates a group without linking
                                        * it into the file structure.     */
status        = H5Olink(obj_id, file_id, "/A/B01/C", lcpl_id, H5P_DEFAULT);
                                       /* Links group into file structure.*/
```

Note that unless the object is intended to be temporary, the `H5Olink` call is mandatory if an object created with one of the `H5*create_anon` functions (or with `H5Tcommit_anon`) is to be retained in the file; without an `H5Olink` call, the object will not be linked into the HDF5 file structure and will be deleted when the file is closed.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5olink_f

```
SUBROUTINE h5olink_f(object_id, new_loc_id, new_link_name, hdferr, &
                     lcpl_id, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: object_id
                                  ! Object to be linked
  INTEGER(HID_T), INTENT(IN) :: new_loc_id
                                  ! File or group identifier specifying
                                  ! location at which object is to be linked.
  CHARACTER(LEN=*), INTENT(IN) :: new_link_name
                                  ! Name of link to be created,
                                  ! relative to new_loc_id.
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
                                  !   Success:  0
                                  !   Failure: -1
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                                  ! Link creation property list identifier.
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                  ! Link creation property list identifier.
END SUBROUTINE h5olink_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oopen

Signature:

hid_t `H5Oopen`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `lapl_id`)

Purpose:

Opens an object in an HDF5 file by location identifier and path name.

Description:

`H5Oopen` opens a group, dataset, or named datatype specified by a location, `loc_id`, and a path name, `name`, in an HDF5 file.

This function opens the object in the same manner as `H5Gopen`, `H5Topen`, and `H5Dopen`. However, `H5Oopen` does not require the type of object to be known beforehand. This can be useful with user-defined links, for instance, when only a path may be known. `H5Oopen` cannot be used to open a dataspace, attribute, property list, or file.

Once an object of unknown type has been opened with `H5Oopen`, the type of that object can be determined by means of an `H5Iget_type` call.

`loc_id` can be either a file or group identifier. `name` must be the path to that object relative to `loc_id`.

`lapl_id` is the link access property list associated with the link pointing to the object. If default link access properties are appropriate, this can be passed in as `H5P_DEFAULT`.

When it is no longer needed, the opened object should be closed with `H5Oclose`, `H5Gclose`, `H5Tclose`, or `H5Dclose`.

Parameters:

hid_t `loc_id`	IN: File or group identifier
const char *`name`	IN: Path to the object, relative to `loc_id`.
hid_t `lapl_id`	IN: Access property list identifier for the link pointing to the object

Returns:

Returns an object identifier for the opened object if successful; otherwise returns a negative value.

Fortran90 Interface: h5oopen_f

```
SUBROUTINE h5oopen_f(loc_id, name, obj_id, hdferr, lapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id  ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Path to the object,
                                        ! relative to loc_id
  INTEGER(HID_T), INTENT(OUT) :: obj_id ! Object identifier for opened object
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        !   Success:  0
                                        !   Failure: -1
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lapl_id
                                        ! Attribute access property list
END SUBROUTINE h5oopen_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oopen_by_addr

Signature:

hid_t `H5Oopen_by_addr`(*hid_t* `loc_id`, *haddr_t* `addr`)

Purpose:

Opens an object using its address within an HDF5 file.

Description:

`H5Oopen_by_addr` opens a group, dataset, or named datatype using its address within an HDF5 file, `addr`. The resulting opened object is identical to an object opened with `H5Oopen` and should be closed with `H5Oclose` or an object-type-specific closing function (such as `H5Gclose`) when no longer needed.

`loc_id` can be either the file identifier or a group identifier in the file. In either case, the HDF5 Library uses the identifier only to identify the file.

The object's address within the file, `addr`, is the byte offset of the first byte of the object header from the beginning of the HDF5 file space, i.e., from the beginning of the super block (see the "HDF5 Storage Model" section of the "The HDF5 Data Model and File Structure" chapter of the *HDF5 User' Guide*).

`addr` can be obtained via either of two function calls. `H5Gget_objinfo` returns the object's address in the `objno` field of the `H5G_stat_t` struct; `H5Lget_linkinfo` returns the address in the `address` field of the `H5L_linkinfo_t` struct.

Warning: This function must be used with care!
Improper use can lead to inaccessible data, wasted space in the file, or ***file corruption***.

This function is dangerous if called on an invalid address. The risk can be safely overcome by retrieving the object address with `H5Gget_objinfo` or `H5Lget_linkinfo` immediately before calling `H5Oopen_by_addr`. The immediacy of the operation can be important; if time has elapsed and the object has been deleted from the file, the address will be invalid and file corruption can result.

The address of the HDF5 file on a physical device has no effect on `H5Oopen_by_addr`, nor does the use of any file driver. As stated above, the object address is its offset within the HDF5 file; HDF5's file drivers will transparently map this to an address on a storage device.

Parameters:

hid_t `loc_id`	IN: File or group identifier
haddr_t `addr`	IN: Object's address in the file

Returns:

Returns an object identifier for the opened object if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oopen_by_idx

Signature:

hid_t `H5Oopen_by_idx`(*hid_t* `loc_id`, *const char* *`group_name`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *hsize_t* `n`, *hid_t* `lapl_id`)

Purpose:

Open the *n*th object in a group.

Description:

`H5Oopen_by_idx` opens the *n*th object in the group specified by `loc_id` and `group_name`.

`loc_id` specifies a file or group. `group_name` specifies the group relative to `loc_id` in which the object can be found. If `loc_id` fully specifies the group in which the object resides, `group_name` can be a dot (.).

The specific object to be opened within the group is specified by `index_type`, `order`, and `n` as follows:

◊ `index_type` specifies the type of index by which objects are ordered. Valid index types include `H5_INDEX_NAME`, indexed by name, and `H5_INDEX_CRT_ORDER`, indexed by creation order.

◊ `order` specifies the order in which the links are to be referenced for the purposes of this function. Valid orders include `H5_ITER_INC` for increasing order, `H5_ITER_DEC` for decreasing order, and `H5_ITER_NATIVE`. Rather than implying a particular order, `H5_ITER_NATIVE` instructs the HDF5 Library to iterate through the objects in the fastest available order, i.e., in a natural order.

◊ `n` specifies the position of the object within the index. Note that this count is zero-based; `0` (zero) indicates that the function will return the value of the first object; if `n` is `5`, the function will return the value of the sixth object; etc.

If `lapl_id` specifies the link access property list to be used in accessing the object.

Parameters:

hid_t `loc_id`	IN: A file or group identifier.
const char *`group_name`	IN: Name of group, relative to `loc_id`, in which object is located
H5_index_t `index_type`	IN: Type of index by which objects are ordered
H5_iter_order_t `order`	IN: Order of iteration within index
hsize_t `n`	IN: Object to open
hid_t `lapl_id`	IN: Link access property list

Returns:

Returns an object identifier for the opened object if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oset_comment

Signature:

herr_t `H5Oset_comment(` *hid_t* `object_id`, *const char* *`comment`)

Purpose:

Sets comment for specified object.

Description:

`H5Oset_comment` sets the comment for the specified object to the contents of `comment`. Any previously existing comment is overwritten.

The target object is specified by an identifier, `object_id`.

If `comment` is the empty string or a null pointer, any existing comment message is removed from the object.

Comments should be relatively short, null-terminated, ASCII strings.

Comments can be attached to any object that has an object header, e.g., datasets, groups, and named datatypes, but not symbolic links.

Parameters:

hid_t `object_id`	IN: Identifier of the target object
const char *`comment`	IN: The new comment.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Oset_comment_by_name

Signature:

herr_t `H5Oset_comment_by_name`(*hid_t* `loc_id`, *const char* *`name`, *const char* *`comment`, *hid_t* `lapl_id`)

Purpose:

Sets comment for specified object.

Description:

`H5Oset_comment_by_name` sets the comment for the specified object to the contents of `comment`. Any previously existing comment is overwritten.

The target object is specified by `loc_id` and `name`. `loc_id` can specify any object in the file. `name` can be one of the following:
— The name of the object relative to `loc_id`
— An absolute name of the object, starting from `/`, the file’s root group
— A dot (`.`), if `loc_id` fully specifies the object

If `comment` is the empty string or a null pointer, any existing comment message is removed from the object.

Comments should be relatively short, null-terminated, ASCII strings.

Comments can be attached to any object that has an object header, e.g., datasets, groups, and named datatypes, but not symbolic links.

`lapl_id` contains a link access property list identifier. A link access propety list can come into play when traversing links to access an object.

Parameters:

hid_t `loc_id`	IN: Identifier of a file, group, dataset, or named datatype.
const char *`name`	IN: Name of the object whose comment is to be set or reset, specified as a path relative to `loc_id`. `name` can be '.' (a dot) if `loc_id` fully specifies the object for which the comment is to be set.
const char *`comment`	IN: The new comment.
hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ovisit

Signature:

herr_t `H5Ovisit`(*hid_t* `object_id`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *H5O_iterate_t* `op`, *void* *`op_data`)

Purpose:

Recursively visits all objects accessible from a specified object.

Description:

`H5Ovisit` is a recursive iteration function to visit the object `object_id` and, if `object_id` is a group, all objects in and below it in an HDF5 file, thus providing a mechanism for an application to perform a common set of operations across all of those objects or a dynamically selected subset. For non-recursive iteration across the members of a group, see `H5Literate`.

If `object_id` is a group identifier, that group serves as the root of a recursive iteration. If `object_id` is a file identifier, that file's root group serves as the root of the recursive iteration. If `object_id` is any other type of object, such as a dataset or named datatype, there is no iteration.

Two parameters are used to establish the iteration: `index_type` and `order`.

`index_type` specifies the index to be used. If the links in a group have not been indexed by the index type, they will first be sorted by that index then the iteration will begin; if the links have been so indexed, the sorting step will be unnecesary, so the iteration may begin more quickly. Valid values include the following:

`H5_INDEX_NAME`	Alpha-numeric index on name
`H5_INDEX_CRT_ORDER`	Index on creation order

Note that the index type passed in `index_type` is a *best effort* setting. If the application passes in a value indicating iteration in creation order and a group is encountered that was not tracked in creation order, that group will be iterated over in alpha-numeric order by name, or *name order*. (*Name order* is the native order used by the HDF5 Library and is always available.)

`order` specifies the order in which objects are to be inspected along the index specified in `index_type`. Valid values include the following:

`H5_ITER_INC`	Increasing order
`H5_ITER_DEC`	Decreasing order
`H5_ITER_NATIVE`	Fastest available order

The protoype of the callback function `op` is as follows (as defined in the source code file `H5Opublic.h`):

herr_t `(*H5O_iterate_t)(` *hid_t* `o_id,` *const char* *`name,` *const H5O_info_t* *`object_info,` *void* *`op_data)`

The parameters of this callback function have the following values or meanings:

`o_id`	Object that serves as root of the iteration; same value as the `H5Ovisit` `object_id` parameter
`name`	

	Name of object, relative to `o_id`, being examined at current step of the iteration
`object_info`	*H5O_info_t* struct containing information regarding that object
`op_data`	User-defined pointer to data required by the application in processing the object; a pass-through of the `op_data` pointer provided with the `H5Ovisit_by_name` function call

The *H5O_info_t* struct is defined in `H5Opublic.h` and described in the `H5Oget_info` function entry.

The return values from an operator are:

◊ Zero causes the visit iterator to continue, returning zero when all group members have been processed.
◊ A positive value causes the visit iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the next group member.
◊ A negative value causes the visit iterator to immediately return that value, indicating failure. The iterator can be restarted at the next group member.

The `H5Ovisit` `op_data` parameter is a user-defined pointer to the data required to process objects in the course of the iteration. This pointer is passed back to each step of the iteration in the callback function's `op_data` parameter.

`H5Lvisit` and `H5Ovisit` are companion functions: one for examining and operating on links; the other for examining and operating on the objects that those links point to. Both functions ensure that by the time the function completes successfully, every link or object below the specified point in the file has been presented to the application for whatever processing the application requires.

Parameters:

hid_t `object_id`	IN: Identifier of the object at which the recursive iteration begins.

H5_index_t `index_type`	IN: Type of index; valid values include:
	`H5_INDEX_NAME`
	`H5_INDEX_CRT_ORDER`

H5_iter_order_t `order`	IN: Order in which index is traversed; valid values include:
	`H5_ITER_DEC`
	`H5_ITER_INC`
	`H5_ITER_NATIVE`

H5O_iterate_t `op`	IN: Callback function passing data regarding the object to the calling application
void *`op_data`	IN: User-defined pointer to data required by the application for its processing of the object

Returns:

On success, returns the return value of the first operator that returns a positive value, or zero if all members were processed with no operator returning non-zero.

On failure, returns a negative value if something goes wrong within the library, or the first negative value returned by an operator.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Ovisit_by_name

Signature:

herr_t `H5Ovisit_by_name`(*hid_t* `loc_id`, *const char* *`object_name`, *H5_index_t* `index_type`, *H5_iter_order_t* `order`, *H5O_iterate_t* `op`, *void* *`op_data`, *hid_t* `lapl_id`)

Purpose:

Recursively visits all objects starting from a specified object.

Description:

`H5Ovisit_by_name` is a recursive iteration function to visit the object specified by the `loc_id` / `object_name` parameter pair and, if that object is a group, all objects in and below it in an HDF5 file, thus providing a mechanism for an application to perform a common set of operations across all of those objects or a dynamically selected subset. For non-recursive iteration across the members of a group, see `H5Literate`.

The object serving as the root of the iteration is specified by the `loc_id` / `object_name` parameter pair. `loc_id` specifies a file or an object in a file; `object_name` specifies either an object in the file (with an absolute name based in the file's root group) or an object name relative to `loc_id`. If `loc_id` fully specifies the object that is to serve as the root of the iteration, `object_name` should be `'.'` (a dot). (Note that when `loc_id` fully specifies the the object that is to serve as the root of the iteration, the user may wish to consider using `H5Ovisit` instead of `H5Ovisit_by_name`.)

Two parameters are used to establish the iteration: `index_type` and `order`.

`index_type` specifies the index to be used. If the links in a group have not been indexed by the index type, they will first be sorted by that index then the iteration will begin; if the links have been so indexed, the sorting step will be unnecesary, so the iteration may begin more quickly. Valid values include the following:

`H5_INDEX_NAME`	Alpha-numeric index on name
`H5_INDEX_CRT_ORDER`	Index on creation order

Note that the index type passed in `index_type` is a *best effort* setting. If the application passes in a value indicating iteration in creation order and a group is encountered that was not tracked in creation order, that group will be iterated over in alpha-numeric order by name, or *name order*. (*Name order* is the native order used by the HDF5 Library and is always available.)

`order` specifies the order in which objects are to be inspected along the index specified in `index_type`. Valid values include the following:

`H5_ITER_INC`	Increasing order
`H5_ITER_DEC`	Decreasing order
`H5_ITER_NATIVE`	Fastest available order

The `op` callback funtion and the effect of the callback function's return value on the application are described in `H5Ovisit`.

The *H5O_info_t* struct is defined in `H5Opublic.h` and described in the `H5Oget_info` function entry.

The `H5Ovisit_by_name` `op_data` parameter is a user-defined pointer to the data required to process objects in the course of the iteration. This pointer is passed back to each step of the iteration in the callback function's `op_data` parameter.

`lapl_id` is a link access property list. In the general case, when default link access properties are acceptable, this can be passed in as `H5P_DEFAULT`. An example of a situation that requires a non-default link access property list is when the link is an external link; an external link may require that a link prefix be set in a link access property list (see `H5Pset_elink_prefix`).

`H5Lvisit_by_name` and `H5Ovisit_by_name` are companion functions: one for examining and operating on links; the other for examining and operating on the objects that those links point to. Both functions ensure that by the time the function completes successfully, every link or object below the specified point in the file has been presented to the application for whatever processing the application requires.

Parameters:

hid_t `loc_id`	IN: Identifier of a file or group
const char *`object_name`	IN: Name of the object, generally relative to `loc_id`, that will serve as root of the iteration

H5_index_t `index_type`	IN: Type of index; valid values include:
	`H5_INDEX_NAME`
	`H5_INDEX_CRT_ORDER`

H5_iter_order_t `order`	IN: Order in which index is traversed; valid values include:
	`H5_ITER_DEC`
	`H5_ITER_INC`
	`H5_ITER_NATIVE`

H5O_iterate_t `op`	IN: Callback function passing data regarding the object to the calling application
void *`op_data`	IN: User-defined pointer to data required by the application for its processing of the object
hid_t `lapl_id`	IN: Link access property list identifier

Returns:

On success, returns the return value of the first operator that returns a positive value, or zero if all members were processed with no operator returning non-zero.

On failure, returns a negative value if something goes wrong within the library, or the first negative value returned by an operator.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

H5P: Property List Interface

Property List API Functions

These functions manipulate property list objects to allow objects which require many different parameters to be easily manipulated.
In the following lists, italic type indicates a configurable macro.

The C Interfaces:

General Functions

- H5Pcreate
- H5Pget_class
- H5Pcopy
- H5Pclose

Generic Properties

- H5Pcreate_class
- *H5Pregister*
- H5Pregister1 *
- H5Pregister2
- *H5Pinsert*
- H5Pinsert1 *
- H5Pinsert2
- H5Pset
- H5Pexist
- H5Pget_size
- H5Pget_nprops
- H5Pget_class_name
- H5Pget_class_parent
- H5Pisa_class
- H5Pget
- H5Pequal
- H5Piterate
- H5Pcopy_prop
- H5Premove
- H5Punregister
- H5Pclose_class

File Access Properties

- H5Pset_driver
- H5Pget_driver_info
- H5Pset_fclose_degree
- H5Pget_fclose_degree
- H5Pset_fapl_core
- H5Pget_fapl_core
- H5Pset_fapl_direct
- H5Pget_fapl_direct
- H5Pset_fapl_family
- H5Pget_fapl_family
- H5Pset_family_offset
- H5Pget_family_offset
- H5Pset_fapl_log
- H5Pset_fapl_mpio ||
- H5Pget_fapl_mpio ||
- H5Pset_fapl_mpiposix ||
- H5Pget_fapl_mpiposix ||
- H5Pset_fapl_multi
- H5Pget_fapl_multi
- H5Pset_multi_type
- H5Pget_multi_type
- H5Pset_fapl_split
- H5Pset_fapl_sec2
- H5Pset_fapl_stdio
- H5Pset_fapl_windows
- H5set_driver
- H5Pget_driver
- H5Pget_driver_info
- H5Pset_meta_block_size
- H5Pget_meta_block_size
- H5Pset_sieve_buf_size
- H5Pget_sieve_buf_size
- H5Pset_alignment
- H5Pget_alignment
- H5Pset_cache
- H5Pget_cache

File Access Properties (cont.)

- H5Pget_mdc_config
- H5Pset_mdc_config
- H5Pset_gc_references
- H5Pget_gc_references
- H5Pset_small_data_block_size
- H5Pget_small_data_block_size
- H5Pset_libver_bounds
- H5Pget_libver_bounds

File Creation Properties

- H5Pget_version
- H5Pset_userblock
- H5Pget_userblock
- H5Pset_sizes
- H5Pget_sizes
- H5Pset_sym_k
- H5Pget_sym_k
- H5Pset_istore_k
- H5Pget_istore_k
- H5Pset_shared_mesg_nindexes
- H5Pget_shared_mesg_nindexes
- H5Pset_shared_mesg_index
- H5Pget_shared_mesg_index
- H5Pset_shared_mesg_phase_change
- H5Pget_shared_mesg_phase_change

|| *Indicates functions available only in the parallel HDF5 library.*

* *Use of these functions is deprecated in Release 1.8.0.*

Dataset Creation Properties

- H5Pset_layout
- H5Pget_layout
- H5Pset_chunk
- H5Pget_chunk
- H5Pset_chunk_cache
- H5Pget_chunk_cache
- H5Pset_deflate
- H5Pset_fill_value
- H5Pget_fill_value
- H5Pfill_value_defined
- H5Pset_fill_time
- H5Pget_fill_time
- H5Pset_alloc_time
- H5Pget_alloc_time
- H5Pset_filter
- H5Pall_filters_avail
- H5Pget_nfilters
- *H5Pget_filter*
- H5Pget_filter1 *
- H5Pget_filter2
- *H5Pget_filter_by_id*
- H5Pget_filter_by_id1 *
- H5Pget_filter_by_id2
- H5Pmodify_filter
- H5Premove_filter
- H5Pset_fletcher32
- H5Pset_shuffle
- H5Pset_szip
- H5Pset_external
- H5Pget_external_count
- H5Pget_external

Group Creation Properties

- H5Pset_create_intermediate_group
- H5Pget_create_intermediate_group
- H5Pset_link_creation_order
- H5Pget_link_creation_order
- H5Pset_est_link_info
- H5Pget_est_link_info
- H5Pset_local_heap_size_hint
- H5Pget_local_heap_size_hint
- H5Pset_link_phase_change
- H5Pget_link_phase_change

Dataset Access, Memory, and Transfer Properties

- H5Pset_buffer
- H5Pget_buffer
- H5Pset_preserve *
- H5Pget_preserve *
- H5Pset_edc_check
- H5Pget_edc_check
- H5Pset_data_transform
- H5Pget_data_transform
- H5Pset_filter_callback
- H5Pset_hyper_vector_size
- H5Pget_hyper_vector_size
- H5Pset_btree_ratios
- H5Pget_btree_ratios
- H5Pset_vlen_mem_manager
- H5Pget_vlen_mem_manager
- H5Pset_dxpl_mpio ||
- H5Pset_dxpl_mpio_chunk_opt ||
- H5Pset_dxpl_mpio_chunk_opt_num ||
- H5Pset_dxpl_mpio_chunk_opt_ratio ||
- H5Pset_dxpl_mpio_collective_opt ||
- H5Pget_dxpl_mpio ||
- H5Pset_dxpl_multi
- H5Pget_dxpl_multi

Object Copy and Object Creation Properties

- H5Pset_create_intermediate_group
- H5Pget_create_intermediate_group
- H5Pset_copy_object
- H5Pget_copy_object
- H5Pset_attr_phase_change
- H5Pget_attr_phase_change
- H5Pset_attr_creation_order
- H5Pget_attr_creation_order
- H5Pset_obj_track_times
- H5Pget_obj_track_times

Link Access Properties

- H5Pset_nlinks
- H5Pget_nlinks
- H5Pset_elink_cb
- H5Pget_elink_cb
- H5Pset_elink_prefix
- H5Pget_elink_prefix
- H5Pset_elink_fapl
- H5Pget_elink_fapl
- H5Pset_elink_acc_flags
- H5Pget_elink_acc_flags

String Properties

- H5Pset_char_encoding
- H5Pget_char_encoding

|| *Indicates functions available only in the parallel HDF5*

* *Use of these functions is deprecated in Release 1.8.0.*

- H5Pall_filters_avail
- H5Pclose
- H5Pclose_class
- H5Pcopy
- H5Pcopy_prop
- H5Pcreate
- H5Pcreate_class
- H5Pequal
- H5Pexist
- H5Pfill_value_defined
- H5Pget
- H5Pget_alignment
- H5Pget_alloc_time
- H5Pget_attr_creation_order
- H5Pget_attr_phase_change
- H5Pget_btree_ratios
- H5Pget_buffer
- H5Pget_cache
- H5Pget_char_encoding
- H5Pget_chunk
- H5Pget_chunk_cache
- H5Pget_class
- H5Pget_class_name
- H5Pget_class_parent
- H5Pget_copy_object
- H5Pget_create_intermediate_group
- H5Pget_data_transform
- H5Pget_driver
- H5Pget_driver_info
- H5Pget_dxpl_mpio ||
- H5Pget_dxpl_multi
- H5Pget_edc_check
- H5Pget_elink_acc_flags
- H5Pget_elink_cb
- H5Pget_elink_fapl
- H5Pget_elink_prefix
- H5Pget_est_link_info
- H5Pget_external
- H5Pget_external_count
- H5Pget_family_offset
- H5Pget_fapl_core
- H5Pget_fapl_direct
- H5Pget_fapl_family
- H5Pget_fapl_mpio ||
- H5Pget_fapl_mpiposix ||
- H5Pget_fapl_multi
- H5Pget_fclose_degree
- H5Pget_fill_time
- H5Pget_fill_value
- *H5Pget_filter*
- H5Pget_filter1 *
- H5Pget_filter2
- *H5Pget_filter_by_id*
- H5Pget_filter_by_id1 *
- H5Pget_filter_by_id2
- H5Pget_gc_references
- H5Pget_hyper_vector_size
- H5Pget_istore_k
- H5Pget_layout
- H5Pget_libver_bounds
- H5Pget_link_creation_order
- H5Pget_link_phase_change
- H5Pget_local_heap_size_hint
- H5Pget_mdc_config
- H5Pget_meta_block_size
- H5Pget_multi_type
- H5Pget_nfilters
- H5Pget_nlinks
- H5Pget_nprops
- H5Pget_preserve *
- H5Pget_obj_track_times
- H5Pget_shared_mesg_index
- H5Pget_shared_mesg_nindexes
- H5Pget_shared_mesg_phase_change
- H5Pget_sieve_buf_size
- H5Pget_size
- H5Pget_sizes
- H5Pget_small_data_block_size
- H5Pget_sym_k
- H5Pget_type_conv_cb
- H5Pget_userblock
- H5Pget_version
- H5Pget_vlen_mem_manager
- *H5Pinsert*
- H5Pinsert1 *
- H5Pinsert2
- H5Pisa_class
- H5Piterate
- H5Pmodify_filter

- *H5Pregister*
- H5Pregister1 *
- H5Pregister2
- H5Premove
- H5Premove_filter
- H5Pset
- H5Pset_alignment
- H5Pset_alloc_time
- H5Pset_attr_creation_order
- H5Pset_attr_phase_change
- H5Pset_btree_ratios
- H5Pset_buffer
- H5Pset_cache
- H5Pset_char_encoding
- H5Pset_chunk
- H5Pset_chunk_cache
- H5Pset_copy_object
- H5Pset_create_intermediate_group
- H5Pset_data_transform
- H5Pset_deflate
- H5Pset_driver
- H5Pset_dxpl_mpio ||
- H5Pset_dxpl_mpio_chunk_opt ||
- H5Pset_dxpl_mpio_chunk_opt_num ||
- H5Pset_dxpl_mpio_chunk_opt_ratio ||
- H5Pset_dxpl_mpio_collective_opt ||
- H5Pset_dxpl_multi
- H5Pset_edc_check
- H5Pset_elink_acc_flags
- H5Pset_elink_cb
- H5Pset_elink_fapl
- H5Pset_elink_prefix
- H5Pset_est_link_info
- H5Pset_external
- H5Pset_family_offset
- H5Pset_fapl_core
- H5Pset_fapl_family
- H5Pset_fapl_direct
- H5Pset_fapl_log
- H5Pset_fapl_mpio ||
- H5Pset_fapl_mpiposix ||
- H5Pset_fapl_multi
- H5Pset_fapl_sec2
- H5Pset_fapl_split
- H5Pset_fapl_stdio
- H5Pset_fapl_windows
- H5Pset_fclose_degree
- H5Pset_fill_time
- H5Pset_fill_value
- H5Pset_filter
- H5Pset_filter_callback
- H5Pset_fletcher32
- H5Pset_gc_references
- H5Pset_hyper_vector_size
- H5Pset_istore_k
- H5Pset_layout
- H5Pset_libver_bounds
- H5Pset_link_creation_order
- H5Pset_link_phase_change
- H5Pset_local_heap_size_hint
- H5Pset_mdc_config
- H5Pset_meta_block_size
- H5Pset_multi_type
- H5Pset_nbit
- H5Pset_nlinks
- H5Pset_preserve *
- H5Pset_obj_track_times
- H5Pset_scaleoffset
- H5Pset_shared_mesg_index
- H5Pset_shared_mesg_nindexes
- H5Pset_shared_mesg_phase_change
- H5Pset_shuffle
- H5Pset_sieve_buf_size
- H5Pset_sizes
- H5Pset_small_data_block_size
- H5Pset_sym_k
- H5Pset_szip
- H5Pset_type_conv_cb
- H5Pset_userblock
- H5Pset_vlen_mem_manager
- H5Punregister

|| *Indicates functions available only in the parallel HDF5 library.*

* *Use of these functions is deprecated in Release 1.8.0.*

The FORTRAN90 Interfaces:

In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

General Property List Operations

- h5pcreate_f
- h5pget_class_f
- h5pcopy_f
- h5pclose_f

Generic Properties

- h5pcreate_class_f
- h5pregister_f
- h5pinsert_f
- h5pset_f
- h5pexist_f
- h5pget_size_f
- h5pget_nprops_f
- h5pget_class_name_f
- h5pget_class_parent_f
- h5pisa_class_f
- h5pget_f
- h5pequal_f
- h5pcopy_prop_f
- h5premove_f
- h5punregister_f
- h5pclose_class_f

File Creation Properties

- H5Pget_version_f
- H5Pset_userblock_f
- H5Pget_userblock_f
- H5Pset_sizes_f
- H5Pget_sizes_f
- H5Pset_sym_k_f
- H5Pget_sym_k_f
- H5Pset_istore_k_f
- H5Pget_istore_k_f

File Access Properties

- H5Pset_driver_f
- H5Pget_driver_info_f
- H5Pset_fclose_degree_f
- H5Pget_fclose_degree_f
- H5Pset_fapl_core_f
- H5Pget_fapl_core_f
- H5Pset_fapl_direct_f
- H5Pget_fapl_direct_f
- H5Pset_fapl_family_f
- H5Pget_fapl_family_f
- H5Pset_family_offset_f
- H5Pset_fapl_mpio_f ||
- H5Pget_fapl_mpio_f ||
- H5Pset_fapl_mpiposix_f ||
- H5Pget_fapl_mpiposix_f ||
- H5Pset_fapl_multi_f
- H5Pget_fapl_multi_f
- H5Pset_multi_type_f
- H5Pget_multi_type_f
- H5Pset_fapl_split_f
- H5Pset_fapl_sec2_f
- H5Pset_fapl_stdio_f
- H5Pget_driver_f
- H5Pset_meta_block_size_f
- H5Pget_meta_block_size_f
- H5Pset_sieve_buf_size_f
- H5Pget_sieve_buf_size_f
- H5Pset_alignment_f
- H5Pget_alignment_f
- H5Pset_cache_f
- H5Pget_cache_f
- H5Pset_gc_references_f
- H5Pget_gc_references_f
- H5Pset_small_data_block_size_f
- H5Pget_small_data_block_size_f
- H5Pset_libver_bounds_f

Group Creation Properties

- H5Pset_create_inter_group_f
- H5Pget_create_inter_group_f
- H5Pset_local_heap_size_hint_f
- H5Pget_local_heap_size_hint_f
- H5Pset_link_creation_order_f
- H5Pget_link_creation_order_f
- H5Pset_est_link_info_f
- H5Pget_est_link_info_f
- H5Pset_link_phase_change_f
- H5Pget_link_phase_change_f

Link Access Properties

- H5Pset_nlinks_f
- H5Pget_nlinks_f

|| *Indicates functions available only in the parallel HDF5 library.*

Object Copy and
Object Creation Properties

- H5Pset_create_inter_group_f
- H5Pget_create_inter_group_f
- H5Pset_copy_object_f
- H5Pget_copy_object_f
- H5Pset_attr_phase_change_f
- H5Pget_attr_phase_change_f
- H5Pset_attr_creation_order_f
- H5Pget_attr_creation_order_f
- H5Pset_obj_track_times_f
- H5Pget_obj_track_times_f

String Properties

- H5Pset_char_encoding_f
- H5Pget_char_encoding_f

Dataset Creation Properties

- H5Pset_layout_f
- H5Pget_layout_f
- H5Pset_chunk_f
- H5Pget_chunk_f
- H5Pset_deflate_f
- H5Pset_fill_value_f
- H5Pget_fill_value_f
- H5Pset_fill_time_f
- H5Pget_fill_time_f
- H5Pset_alloc_time_f
- H5Pget_alloc_time_f
- H5Pset_filter_f
- H5Pget_nfilters_f
- H5Pget_filter_f
- H5Pget_filter_by_id_f
- H5Pmodify_filter_f
- H5Premove_filter_f
- H5Pset_fletcher32_f
- H5Pset_nbit_f
- H5Pset_scaleoffset_f
- H5Pset_shuffle_f
- H5Pset_szip_f
- H5Pset_external_f
- H5Pget_external_count_f
- H5Pget_external_f

Dataset Access, Memory,
and
Transfer Properties

- H5Pset_buffer_f
- H5Pget_buffer_f
- H5Pset_preserve_f
- H5Pget_preserve_f
- H5Pset_edc_check_f
- H5Pget_edc_check_f
- H5Pset_data_transform_f
- H5Pget_data_transform_f
- H5Pset_hyper_vector_size_f
- H5Pget_hyper_vector_size_f
- H5Pset_btree_ratios_f
- H5Pget_btree_ratios_f
- H5Pset_dxpl_mpio_f ||
- H5Pget_dxpl_mpio_f ||

|| *Indicates functions available only in the parallel HDF5 library.*

* *Use of these functions is deprecated in Release 1.8.0.*

Filter Behavior in HDF5:
Filters can be inserted into the HDF5 pipeline to perform functions such as compression and conversion. As such, they are a very flexible aspect of HDF5; for example, a user-defined filter could provide encryption for an HDF5 dataset.

A filter can be declared as either *required* or *optional*. Required is the default status; optional status must be explicitly declared.

A required filter that fails or is not defined causes an entire output operation to fail; if it was applied when the data was written, such a filter will cause an input operation to fail.

The following table summarizes required filter behavior.

	Required FILTER_X not available	**FILTER_X available**
`H5Pset_<FILTER_X>`	Will fail.	Will succeed.
`H5Dwrite` **with FILTER_X set**	Will fail.	Will succeed; FILTER_X will be applied to the data.
`H5Dread` **with FILTER_X set**	Will fail.	Will succeed.

An optional filter can be set for an HDF5 dataset even when the filter is not available. Such a filter can then be applied to the dataset when it becomes available on the original system or when the file containing the dataset is processed on a system on which it is available.

A filter can be declared as optional through the use of the `H5Z_FLAG_OPTIONAL` flag with `H5Pset_filter`.

Consider a situation where one is creating files that will normally be used only on systems where the optional (and fictional) filter FILTER_Z is routinely available. One can create those files on system A, which lacks FILTER_Z, create chunked datasets in the files with FILTER_Z defined in the dataset creation property list, and even write data to those datasets. The dataset object header will indicate that FILTER_Z has been associated with this dataset. But since system A does not have FILTER_Z, dataset chunks will be written without it being applied.

HDF5 has a mechanism for determining whether chunks are actually written with the filters specified in the object header, so while the filter remains unavailable, system A will be able to read the data. Once the file is moved to system B, where FILTER_Z *is* available, HDF5 will apply FILTER_Z to any data rewritten or new data written in these datasets. Dataset chunks that have been written on system B will then be unreadable on system A; chunks that have not been re-written since being written on system A will remain readable on system A. All chunks will be readable on system B.

The following table summarizes optional filter behavior.

	FILTER_Z not available	**FILTER_Z available with encode and decode**	**FILTER_Z available decode only**
`H5Pset_<FILTER_Z>`	Will succeed.	Will succeed.	Will succeed.
`H5Dwrite` **with FILTER_Z set**	Will succeed; FILTER_Z will *not* be applied to the data.	Will succeed; FILTER_Z will be applied to the data.	Will succeed; FILTER_Z will *not* be applied to the data.
`H5Dread` **with FILTER_Z set**	Will succeed if FILTER_Z has not actually been applied to data.	Will succeed.	Will succeed.

The above principles apply generally in the use of HDF5 optional filters insofar as HDF5 does as much as possible to complete an operation when an optional filter is unavailable. (The SZIP filter is an exception to this rule; see `H5Pset_szip` for details.)

Notes:
Filters can be applied only to chunked datasets; they cannot be used with other dataset storage methods, such as contiguous, compact, or external datasets.

Dataset elements of variable-length and dataset region reference datatypes are stored in separate structures in the file called heaps. Filters cannot currently be applied to these heaps.

Name: H5Pall_filters_avail

Signature:

htri_t `H5Pall_filters_avail`(*hid_t* `dcpl_id`)

Purpose:

Verifies that all required filters are available.

Description:

`H5Pall_filters_avail` verifies that all of the filters set in the dataset creation property list `dcpl_id` are currently available.

Parameters:

hid_t `dcpl_id`	IN: Dataset creation property list identifier.

Returns:

Returns `TRUE` if all filters are available and `FALSE` if one or more is not currently available. Returns `FAIL`, a negative value, on error.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pclose

Signature:

herr_t `H5Pclose`(*hid_t* `plist`)

Purpose:

Terminates access to a property list.

Description:

`H5Pclose` terminates access to a property list. All property lists should be closed when the application is finished accessing them. This frees resources used by the property list.

Parameters:

hid_t `plist`	IN: Identifier of the property list to terminate access to.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pclose_f

```
SUBROUTINE h5pclose_f(prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pclose_f
```

Name: H5Pclose_class

Signature:

herr_t `H5Pclose_class`(*hid_t* `class`)

Purpose:

Closes an existing property list class.

Description:

Removes a property list class from the library.

Existing property lists of this class will continue to exist, but new ones are not able to be created.

Parameters:

hid_t `class`	IN: Property list class to close

Returns:

Success: a non-negative value

Failure: a negative value

Fortran90 Interface: h5pclose_class_f

```
SUBROUTINE h5pclose_class_f(class, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: class ! Property list class identifier
                                      ! to close
  INTEGER, INTENT(OUT) :: hdferr      ! Error code
                                      ! 0 on success and -1 on failure
END SUBROUTINE h5pclose_class_f
```

Name: H5Pcopy

Signature:

hid_t `H5Pcopy`(*hid_t* `plist`)

Purpose:

Copies an existing property list to create a new property list.

Description:

`H5Pcopy` copies an existing property list to create a new property list. The new property list has the same properties and values as the original property list.

Parameters:

hid_t `plist`	IN: Identifier of property list to duplicate.

Returns:

Returns a property list identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5pcopy_f

```
SUBROUTINE h5pcopy_f(prp_id, new_prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id       ! Property list identifier
  INTEGER(HID_T), INTENT(OUT) :: new_prp_id  ! Identifier  of property list
                                             ! copy
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5pcopy_f
```

Name: H5Pcopy_prop

Signature:

herr_t `H5Pcopy_prop`(*hid_t* `dst_id`, *hid_t* `src_id`, *const char* `*name`)

Purpose:

Copies a property from one list or class to another.

Description:

`H5Pcopy_prop` copies a property from one property list or class to another.

If a property is copied from one class to another, all the property information will be first deleted from the destination class and then the property information will be copied from the source class into the destination class.

If a property is copied from one list to another, the property will be first deleted from the destination list (generating a call to the `close` callback for the property, if one exists) and then the property is copied from the source list to the destination list (generating a call to the `copy` callback for the property, if one exists).

If the property does not exist in the class or list, this call is equivalent to calling `H5Pregister` or `H5Pinsert` (for a class or list, as appropriate) and the `create` callback will be called in the case of the property being copied into a list (if such a callback exists for the property).

Parameters:

hid_t `dst_id`	IN: Identifier of the destination property list or class
hid_t `src_id`	IN: Identifier of the source property list or class
const char `*name`	IN: Name of the property to copy

Returns:

Success: a non-negative value

Failure: a negative value

Fortran90 Interface: h5pcopy_prop_f

```
SUBROUTINE h5pcopy_prop_f(dst_id, src_id, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dst_id  ! Destination property list
                                        ! identifier
  INTEGER(HID_T), INTENT(IN) :: src_id  ! Source property list identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Property name
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pcopy_prop_f
```

Name: H5Pcreate

Signature:

hid_t `H5Pcreate`(*hid_t* `cls_id`)

Purpose:

Creates a new property as an instance of a property list class.

Description:

`H5Pcreate` creates a new property as an instance of some property list class. The new property list is initialized with default values for the specified class. The classes are as follows; see the function index at the top of this page for a list of functions related to each class:

`H5P_OBJECT_CREATE`
: Properties for object creation during the object copying process.

`H5P_FILE_CREATE`
: Properties for file creation.

`H5P_FILE_ACCESS`
: Properties for file access.

`H5P_DATASET_CREATE`
: Properties for dataset creation.

`H5P_DATASET_ACCESS`
: Properties for dataset access.

`H5P_DATASET_XFER`
: Properties for raw data transfer.

`H5P_FILE_MOUNT`
: Properties for file mounting.

`H5P_GROUP_CREATE`
: Properties for group creation during the object copying process.

`H5P_GROUP_ACCESS`
: Properties for group access during the object copying process.

`H5P_DATATYPE_CREATE`
: Properties for datatype creation during the object copying process.

`H5P_DATATYPE_ACCESS`
: Properties for datatype access during the object copying process.

`H5P_STRING_CREATE`
: Properties for character encoding when encoding strings or object names.

`H5P_ATTRIBUTE_CREATE`
: Properties for attribute creation during the object copying process.

`H5P_OBJECT_COPY`
: Properties governing the object copying process.

`H5P_LINK_CREATE`
: Properties governing link creation.

`H5P_LINK_ACCESS`
: Properties governing link traversal when accessing objects.

This property list must eventually be closed with `H5Pclose`; otherwise, errors are likely to occur.

Parameters:

hid_t `cls_id`	IN: The class of the property list to create.

Returns:

Returns a property list identifier (`plist`) if successful; otherwise Fail (-1).

Fortran90 Interface: h5pcreate_f

```
SUBROUTINE h5pcreate_f(classtype, prp_id, hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: classtype       ! The type of the property list
                                         ! to be created
                                         ! Possible values are:
                                         !    H5P_FILE_CREATE_F
                                         !    H5P_FILE_ACCESS_F
                                         !    H5P_DATASET_CREATE_F
                                         !    H5P_DATASET_XFER_F
                                         !    H5P_MOUNT_F
  INTEGER(HID_T), INTENT(OUT) :: prp_id  ! Property list identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pcreate_f
```

Last modified: 26 March 2009

Name: H5Pcreate_class

Signature:

hid_t `H5Pcreate_class`(*hid_t* `parent_class`, *const char* *`name`, *H5P_cls_create_func_t* `create`, *void* *`create_data`, *H5P_cls_copy_func_t* `copy`, *void* *`copy_data`, *H5P_cls_close_func_t* `close`, *void* *`close_data`)

Purpose:

Creates a new property list class.

Description:

`H5Pcreate_class` registers a new property list class with the library. The new property list class can inherit from an existing property list class, `parent_class`, or may be derived from the default “empty” class, `NULL`. New classes with inherited properties from existing classes may not remove those existing properties, only add or remove their own class properties. Property list classes defined and supported in the HDF5 Library distribution are listed and briefly described in `H5Pcreate`.

The `create` routine is called when a new property list of this class is being created. The `H5P_cls_create_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_cls_create_func_t`)(*hid_t* `prop_id`, *void* * `create_data`); The parameters to this callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being created
void * `create_data`	IN: User pointer to any *class creation* data required

The `create` routine is called after any registered `create` function is called for each property value. If the `create` routine returns a negative value, the new list is not returned to the user and the property list creation routine returns an error value.

The `copy` routine is called when an existing property list of this class is copied. The `H5P_cls_copy_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_cls_copy_func_t`)(*hid_t* `prop_id`, *void* * `copy_data`); The parameters to this callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list created by copying
void * `copy_data`	IN: User pointer to any *class copy* data required

The `copy` routine is called after any registered `copy` function is called for each property value. If the `copy` routine returns a negative value, the new list is not returned to the user and the property list copy routine returns an error value.

The `close` routine is called when a property list of this class is being closed. The `H5P_cls_close_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_cls_close_func_t`)(*hid_t* `prop_id`, *void* * `close_data`); The parameters to this callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being closed
void * `close_data`	IN: User pointer to any *class close* data required

The `close` routine is called before any registered `close` function is called for each property value. If the `close` routine returns a negative value, the property list close routine returns an error value but the property list is still closed.

Parameters:

hid_t `parent_class`	IN: Property list class to inherit from or `NULL`
const char `*name`	IN: Name of property list class to register
H5P_cls_create_func_t `create`	IN: Callback routine called when a property list is created
void `*create_data`	IN: Pointer to user-defined *class create* data, to be passed along to *class create* callback
H5P_cls_copy_func_t `copy`	IN: Callback routine called when a property list is copied
void `*copy_data`	IN: Pointer to user-defined *class copy* data, to be passed along to *class copy* callback
H5P_cls_close_func_t `close`	IN: Callback routine called when a property list is being closed
void `*close_data`	IN: Pointer to user-defined *class close* data, to be passed along to *class close* callback

Returns:

On success, returns a valid property list class identifier; otherwise returns a negative value.

Fortran90 Interface: h5pcreate_class_f

```
SUBROUTINE h5pcreate_class_f(parent, name, class, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: parent  ! Parent property list class
                                        ! identifier
                                        ! Possible values include:
                                        !    H5P_NO_CLASS_F
                                        !    H5P_FILE_CREATE_F
                                        !    H5P_FILE_ACCESS_F
                                        !    H5P_DATASET_CREATE_F
                                        !    H5P_DATASET_XFER_F
                                        !    H5P_MOUNT_F
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to create
  INTEGER(HID_T), INTENT(OUT) :: class  ! Property list class identifier
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pcreate_class_f
```

Name: H5Pequal

Signature:

htri_t `H5Pequal`(*hid_t* `id1`, *hid_t* `id2`)

Purpose:

Compares two property lists or classes for equality.

Description:

`H5Pequal` compares two property lists or classes to determine whether they are equal to one another.

Either both `id1` and `id2` must be property lists or both must be classes; comparing a list to a class is an error.

Parameters:

hid_t `id1`	IN: First property object to be compared
hid_t `id2`	IN: Second property object to be compared

Returns:

Success: TRUE (positive) if equal; FALSE (zero) if unequal
Failure: a negative value

Fortran90 Interface: h5pequal_f

```
SUBROUTINE h5pequal_f(plist1_id, plist2_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist1_id ! Property list identifier
  INTEGER(HID_T), INTENT(IN) :: plist2_id ! Property list identifier
  LOGICAL, INTENET(OUT)      :: flag      ! Flag
                                          !    .TRUE. if lists are equal
                                          !    .FALSE. otherwise
  INTEGER, INTENT(OUT)       :: hdferr    ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pequal_f
```

Name: H5Pexist
Signature:
htri_t `H5Pexist(` *hid_t* `id`, *const char* `*name`)
Purpose:
Queries whether a property name exists in a property list or class.
Description:
`H5Pexist` determines whether a property exists within a property list or class.
Parameters:

hid_t `id`	IN: Identifier for the property to query
const char `*name`	IN: Name of property to check for

Returns:
Success: a positive value if the property exists in the property object; zero if the property does not exist
Failure: a negative value
Fortran90 Interface: h5pexist_f

```
SUBROUTINE h5pexist_f(prp_id, name, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to modify
  LOGICAL, INTENT(OUT) :: flag          ! Logical flag
                                        !    .TRUE. if exists
                                        !    .FALSE. otherwise
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pexist_f
```

Name: H5Pfill_value_defined

Signature:

herr_t `H5Pfill_value_defined`(*hid_t* `plist_id`, *H5D_fill_value_t* *`status`)

Purpose:

Determines whether fill value is defined.

Description:

`H5Pfill_value_defined` determines whether a fill value is defined in the dataset creation property list `plist_id`.

Valid values returned in `status` are as follows:

`H5D_FILL_VALUE_UNDEFINED`	Fill value is undefined.
`H5D_FILL_VALUE_DEFAULT`	Fill value is the library default.
`H5D_FILL_VALUE_USER_DEFINED`	Fill value is defined by the application.

Note:

`H5Pfill_value_defined` is designed for use in concert with the dataset fill value properties functions `H5Pget_fill_value` and `H5Pget_fill_time`.

See H5Dcreate for further cross-references.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
H5D_fill_value_t *`status`	OUT: Status of fill value in property list.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pget

Signature:

herr_t `H5Pget`(*hid_t* `plid`, *const char* *`name`, *void* *`value`)

Purpose:

Queries the value of a property.

Description:

`H5Pget` retrieves a copy of the value for a property in a property list. If there is a `get` callback routine registered for this property, the copy of the value of the property will first be passed to that routine and any changes to the copy of the value will be used when returning the property value from this routine.

This routine may be called for zero-sized properties with the `value` set to NULL. The `get` routine will be called with a NULL value if the callback exists.

The property name must exist or this routine will fail.

If the `get` callback routine returns an error, `value` will not be modified.

Parameters:

hid_t `plid`	IN: Identifier of the property list to query
const char *`name`	IN: Name of property to query
void *`value`	OUT: Pointer to a location to which to copy the value of of the property

Returns:

Success: a non-negative value

Failure: a negative value

Fortran90 Interface: h5pget_f

```
SUBROUTINE h5pget_f(plid, name, value, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plid    ! Property list identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to get
  TYPE,  INTENT(OUT) :: value           ! Property value
                                        ! Supported types are:
                                        !    INTEGER
                                        !    REAL
                                        !    DOUBLE PRECISION
                                        !    CHARACTER(LEN=*)
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pget_f
```

Name: H5Pget_alignment

Signature:

herr_t `H5Pget_alignment`(*hid_t* `plist`, *hsize_t* `*threshold`, *hsize_t* `*alignment`)

Purpose:

Retrieves the current settings for alignment properties from a file access property list.

Description:

`H5Pget_alignment` retrieves the current settings for alignment properties from a file access property list. The `threshold` and/or `alignment` pointers may be null pointers (NULL).

Parameters:

hid_t `plist`	IN: Identifier of a file access property list.
hsize_t `*threshold`	OUT: Pointer to location of return threshold value.
hsize_t `*alignment`	OUT: Pointer to location of return alignment value.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_alignment_f

```
SUBROUTINE h5pget_alignment_f(prp_id, threshold,  alignment, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id        ! Property list identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: threshold  ! Threshold value
  INTEGER(HSIZE_T), INTENT(OUT) :: alignment  ! Alignment value
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure
END SUBROUTINE h5pget_alignment_f
```

Name: H5Pget_alloc_time

Signature:

herr_t `H5Pget_alloc_time`(*hid_t* `plist_id`, *H5D_alloc_time_t* *`alloc_time`)

Purpose:

Retrieves the timing for storage space allocation.

Description:

`H5Pget_alloc_time` retrieves the timing for allocating storage space for a dataset's raw data. This property is set in the dataset creation property list `plist_id`.

The timing setting is returned in `alloc_time` as one of the following values:

`H5D_ALLOC_TIME_DEFAULT`	Uses the default allocation time, based on the dataset storage method. See the `alloc_time` description in H5Pset_alloc_time for default allocation times for various storage methods.
`H5D_ALLOC_TIME_EARLY`	All space is allocated when the dataset is created.
`H5D_ALLOC_TIME_INCR`	Space is allocated incrementally as data is written to the dataset.
`H5D_ALLOC_TIME_LATE`	All space is allocated when data is first written to the dataset.

Note:

`H5Pget_alloc_time` is designed to work in concert with the dataset fill value and fill value write time properties, set with the functions `H5Pget_fill_value` and `H5Pget_fill_time`.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
H5D_alloc_time_t *`alloc_time`	IN: When to allocate dataset storage space.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_alloc_time_f

```
SUBROUTINE h5pget_alloc_time_f(plist_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id   ! Dataset creation
                                           ! property list identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: flag    ! Allocation time flag
                                           ! Possible values are:
                                           !    H5D_ALLOC_TIME_ERROR_F
                                           !    H5D_ALLOC_TIME_DEFAULT_F
                                           !    H5D_ALLOC_TIME_EARLY_F
                                           !    H5D_ALLOC_TIME_LATE_F
                                           !    H5D_ALLOC_TIME_INCR_F
  INTEGER, INTENT(OUT)       :: hdferr     ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pget_alloc_time_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pget_attr_creation_order

Signature:

herr_t `H5Pget_attr_creation_order(` *hid_t* `ocpl_id`, *unsigned* `*crt_order_flags`)

Purpose:

Retrieves tracking and indexing settings for attribute creation order.

Description:

`H5Pget_attr_creation_order` retrieves the settings for tracking and indexing attribute creation order on an object.

`ocpl_id` is a dataset or group creation property list identifier. The term `ocpl`, for object creation property list, is used when different types of objects may be involved.

`crt_order_flags` returns flags with the following meanings:

`H5P_CRT_ORDER_TRACKED`	Attribute creation order is tracked but not necessarily indexed.
`H5P_CRT_ORDER_INDEXED`	Attribute creation order is indexed (requires `H5P_CRT_ORDER_TRACKED`).

If `crt_order_flags` is returned with a value of `0` (zero), attribute creation order is neither tracked nor indexed.

Parameters:

hid_t `ocpl_id`	IN: Object (group or dataset) creation property list identifier
unsigned `*crt_order_flags`	OUT: Flags specifying whether to track and index attribute creation order

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_attr_creation_order_f

```
SUBROUTINE h5pget_attr_creation_order_f(ocpl_id, crt_order_flags, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: ocpl_id   ! Object (group or dataset) creation
                                          ! property list identifier
  INTEGER, INTENT(OUT) :: crt_order_flags ! Flags specifying whether to track
                                          ! and index attribute creation order
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_attr_creation_order_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_attr_phase_change

Signature:

herr_t `H5Pget_attr_phase_change`(*hid_t* `ocpl_id`, *unsigned* `*max_compact`, *unsigned* `*min_dense`)

Purpose:

Retrieves attribute storage phase change thresholds.

Description:

`H5Pget_attr_phase_change` retrieves threshold values for attribute storage on an object. These thresholds determine the point at which attribute storage changes from compact storage (i.e., storage in the object header) to dense storage (i.e., storage in a heap and indexed with a B-tree).

In the general case, attributes are initially kept in compact storage. When the number of attributes exceeds `max_compact`, attribute storage switches to dense storage. If the number of attributes subsequently falls below `min_dense`, the attributes are returned to compact storage.

If `max_compact` is set to `0` (zero), dense storage always used.

`ocpl_id` is a dataset or group creation property list identifier. The term `ocpl`, for object creation property list, is used when different types of objects may be involved.

Parameters:

hid_t `ocpl_id`	IN: Object (dataset or group) creation property list identifier
unsigned `*max_compact`	OUT: Maximum number of attributes to be stored in compact storage *(Default: 8)*
unsigned `*min_dense`	OUT: Minimum number of attributes to be stored in dense storage *(Default: 6)*

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_attr_phase_change_f

```
SUBROUTINE h5pget_attr_phase_change_f(ocpl_id, max_compact, min_dense, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: ocpl_id
                  ! Object (dataset or group) creation property list identifier
  INTEGER, INTENT(OUT) :: max_compact
                  ! Maximum number of attributes to be stored in compact storage
                  ! (Default: 8)
  INTEGER, INTENT(OUT) :: min_dense
                  ! Minimum number of attributes to be stored in dense storage
                  ! (Default: 6)
  INTEGER, INTENT(OUT) :: hdferr
                  ! Error code:
                  ! 0 on success and -1 on failure
END SUBROUTINE h5pget_attr_phase_change_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_btree_ratios

Signature:

herr_t `H5Pget_btree_ratios`(*hid_t* `plist`, *double* `*left`, *double* `*middle`, *double* `*right`)

Purpose:

Gets B-tree split ratios for a dataset transfer property list.

Description:

`H5Pget_btree_ratios` returns the B-tree split ratios for a dataset transfer property list.

The B-tree split ratios are returned through the non-`NULL` arguments `left`, `middle`, and `right`, as set by the H5Pset_btree_ratios function.

Parameters:

hid_t `plist`	IN: The dataset transfer property list identifier.
double `left`	OUT: The B-tree split ratio for left-most nodes.
double `right`	OUT: The B-tree split ratio for right-most nodes and lone nodes.
double `middle`	OUT: The B-tree split ratio for all other nodes.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_btree_ratios_f

```
SUBROUTINE h5pget_btree_ratios_f(prp_id, left, middle, right, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id
                                   ! Property list identifier
  REAL, INTENT(OUT) :: left        ! B-tree split ratio for left-most nodes
  REAL, INTENT(OUT) :: middle      ! B-tree split ratio for all other nodes
  REAL, INTENT(OUT) :: right       ! The B-tree split ratio for right-most
                                   ! nodes and lone nodes.
  INTEGER, INTENT(OUT) :: hdferr  ! Error code:
                                   ! 0 on success and -1 on failure
END SUBROUTINE h5pget_btree_ratios_f
```

Name: H5Pget_buffer

Signature:

hsize_t `H5Pget_buffer`(*hid_t* `plist`, *void* `**tconv`, *void* `**bkg`)

Purpose:

Reads buffer settings.

Description:

`H5Pget_buffer` reads values previously set with H5Pset_buffer.

Parameters:

hid_t `plist`	IN: Identifier for the dataset transfer property list.
void `**tconv`	OUT: Address of the pointer to application-allocated type conversion buffer.
void `**bkg`	OUT: Address of the pointer to application-allocated background buffer.

Returns:

Returns buffer size, in bytes, if successful; otherwise 0 on failure.

Fortran90 Interface: h5pget_buffer_f

```
SUBROUTINE h5pget_buffer_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)    :: plist_id ! Dataset transfer
                                            ! property list identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: size     ! Conversion buffer size
  INTEGER, INTENT(OUT)          :: hdferr   ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5pget_buffer_f
```

History:

Release	**C**
1.6.0	The return type changed from *hsize_t* to *size_t*.
1.4.0	The return type changed to *hsize_t*.

Name: H5Pget_cache

Signature:

herr_t `H5Pget_cache`(*hid_t* `plist_id`, *int* `*mdc_nelmts`, *size_t* `*rdcc_nelmts`, *size_t* `*rdcc_nbytes`, *double* `*rdcc_w0`)

Purpose:

Queries the raw data chunk cache parameters.

Description:

`H5Pget_cache` retrieves the maximum possible number of elements in the raw data chunk cache, the maximum possible number of bytes in the raw data chunk cache, and the preemption policy value.

Any (or all) arguments may be null pointers, in which case the corresponding datum is not returned.

Note that the `*mdc_nelmts` parameter is not longer used.

Parameters:

hid_t `plist_id`	IN: Identifier of the file access property list.
int `*mdc_nelmts`	IN/OUT: *No longer used.*
size_t `*rdcc_nelmts`	IN/OUT: Number of elements (objects) in the raw data chunk cache.
size_t `*rdcc_nbytes`	IN/OUT: Total size of the raw data chunk cache, in bytes.
double `*rdcc_w0`	IN/OUT: Preemption policy.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_cache_f

```
SUBROUTINE h5pget_cache_f(prp_id, mdc_nelmts, rdcc_nelmts, rdcc_nbytes,
                          rdcc_w0, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id         ! Property list identifier
  INTEGER, INTENT(OUT) :: mdc_nelmts           ! Number of elements (objects)
                                               ! in the meta data cache
  INTEGER(SIZE_T), INTENT(OUT) :: rdcc_nelmts  ! Number of elements (objects)
                                               ! in the meta data cache
  INTEGER(SIZE_T), INTENT(OUT) :: rdcc_nbytes  ! Total size of the raw data
                                               ! chunk cache, in bytes
  REAL, INTENT(OUT) :: rdcc_w0                 ! Preemption policy
  INTEGER, INTENT(OUT) :: hdferr               ! Error code
                                               ! 0 on success and -1 on failure
END SUBROUTINE h5pget_cache_f
```

History:

Release	**C**
1.8.0	Use of the `mdc_nelmts` parameter discontinued. Metadata cache configuration is managed with H5Pset_mdc_config and H5Pget_mdc_config.
1.6.0	The `rdcc_nbytes` and `rdcc_nelmts` parameters changed from type *int* to *size_t*.

Name: H5Pget_char_encoding

Signature:

herr_t `H5Pget_char_encoding`(*hid_t* `plist_id`, *H5T_cset_t* `encoding`)

Purpose:

Retrieves the character encoding used to create a string.

Description:

`H5Pget_char_encoding` retrieves the character encoding used to encode strings or object names that are created with the property list `plist_id`.

Valid values for `encoding` are defined in `H5Tpublic.h` and include the following:

`H5T_CSET_ASCII`	US ASCII
`H5T_CSET_UTF8`	UTF-8 Unicode encoding

Parameters:

hid_t `plist_id`	IN: Property list identifier
H5T_cset_t `encoding`	OUT: String encoding character set

Returns:

Returns a non-negative valule if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_char_encoding_f

```
SUBROUTINE h5pget_char_encoding_f(plist_id, encoding, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id
                                   ! Property list identifier
  INTEGER, INTENT(OUT) :: encoding ! Valid values for encoding are:
                                   !  H5T_CSET_ASCII_F -> US ASCII
                                   !  H5T_CSET_UTF8_F -> UTF-8 Unicode encoding
  INTEGER, INTENT(OUT) :: hdferr   ! Error code:
                                   ! 0 on success and -1 on failure
END SUBROUTINE h5pget_char_encoding_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_chunk

Signature:

int `H5Pget_chunk`(*hid_t* `plist`, *int* `max_ndims`, *hsize_t* * `dims`)

Purpose:

Retrieves the size of chunks for the raw data of a chunked layout dataset.

Description:

`H5Pget_chunk` retrieves the size of chunks for the raw data of a chunked layout dataset. This function is only valid for dataset creation property lists. At most, `max_ndims` elements of `dims` will be initialized.

Parameters:

hid_t `plist`	IN: Identifier of property list to query.
int `max_ndims`	IN: Size of the `dims` array.
hsize_t * `dims`	OUT: Array to store the chunk dimensions.

Returns:

Returns chunk dimensionality if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_chunk_f

```
SUBROUTINE h5pget_chunk_f(prp_id, ndims, dims, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(IN) :: ndims          ! Number of chunk dimensions
                                        ! to return
  INTEGER(HSIZE_T), DIMENSION(ndims), INTENT(OUT) :: dims
                                        ! Array containing sizes of
                                        ! chunk dimensions
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! chunk rank on success
                                        ! and -1 on failure
END SUBROUTINE h5pget_chunk_f
```

Last modified: 8 May 2009

Name: H5Pget_chunk_cache

Signature:

herr_t `H5Pget_chunk_cache(` *hid_t* `dapl_id`, *size_t* `*rdcc_nslots`, *size_t* `*rdcc_nbytes`, *double* `*rdcc_w0`)

Purpose:

Retrieves the raw data chunk cache parameters.

Description:

`H5Pget_chunk_cache` retrieves the number of chunk slots in the raw data chunk cache hash table , the maximum possible number of bytes in the raw data chunk cache, and the preemption policy value.

These values are retrieved from a dataset access property list. If the values have not been set on the property list, then values returned will be the corresponding values from a default file access property list.

Any (or all) pointer arguments may be null pointers, in which case the corresponding datua is not returned.

Parameters:

hid_t `plist_id`	IN: Dataset access property list identifier.
size_t `*rdcc_nslots`	OUT: Number of chunk slots in the raw data chunk cache hash table.
size_t `*rdcc_nbytes`	OUT: Total size of the raw data chunk cache, in bytes.
double `*rdcc_w0`	OUT: Preemption policy.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The following code retrieves the chunk cache settings on the dataset access property list `dapl_id` into local variables:

```
size_t nslots, nbytes;
double w0;
status = H5Pget_chunk_cache(dapl_id, &nslots, &nbytes, &w0);
```

See Also:

`H5Pset_chunk_cache`

History:

Release	**Change**
1.8.3	C function introduced in this release.

Name: H5Pget_class
Signature:

H5P_class_t `H5Pget_class`(*hid_t* `plist`)

Purpose:

Returns the property list class for a property list.

Description:

`H5Pget_class` returns the property list class for the property list identified by the `plist` parameter. Valid property list classes are defined in the description of `H5Pcreate`.

Parameters:

hid_t `plist`	IN: Identifier of property list to query.

Returns:

Returns a property list class if successful. Otherwise returns H5P_NO_CLASS (-1).

Fortran90 Interface: h5pget_class_f

```
SUBROUTINE h5pget_class_f(prp_id, classtype, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(OUT) :: classtype    ! The type of the property list
                                       ! to be created
                                       ! Possible values are:
                                       !    H5P_NO_CLASS
                                       !    H5P_FILE_CREATE_F
                                       !    H5P_FILE_ACCESS_F
                                       !    H5PE_DATASET_CREATE_F
                                       !    H5P_DATASET_XFER_F
                                       !    H5P_MOUNT_F
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pget_class_f
```

Name: H5Pget_class_name

Purpose:

Retrieves the name of a class.

Signature:

char * `H5Pget_class_name(` *hid_t* `pcid`)

Description:

`H5Pget_class_name` retrieves the name of a generic property list class. The pointer to the name must be freed by the user after each successful call.

Parameters:

hid_t `pcid`	IN: Identifier of the property class to query

Returns:

Success: a pointer to an allocated string containing the class name

Failure: NULL

Fortran90 Interface: h5pget_class_name_f

```
SUBROUTINE h5pget_class_name_f(prp_id, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id     ! Property list identifier to
                                           ! query
  CHARACTER(LEN=*), INTENT(INOUT) :: name  ! Buffer to retrieve class name
  INTEGER, INTENT(OUT) :: hdferr           ! Error code, possible values:
                                           ! Success:  Actual length of the
                                           ! class name
                                           ! If provided buffer "name" is
                                           ! smaller, than name will be
                                           ! truncated to fit into
                                           ! provided user buffer
                                           ! Failure: -1
END SUBROUTINE h5pget_class_name_f
```

Name: H5Pget_class_parent

Signature:

hid_t `H5Pget_class_parent(` *hid_t* `pcid`)

Purpose:

Retrieves the parent class of a property class.

Description:

`H5Pget_class_parent` retrieves an identifier for the parent class of a property class.

Parameters:

hid_t `pcid`	IN: Identifier of the property class to query

Returns:

Success: a valid parent class object identifier

Failure: a negative value

Fortran90 Interface: h5pget_class_parent_f

```
SUBROUTINE h5pget_class_parent_f(prp_id, parent_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id     ! Property list identifier
  INTEGER(HID_T), INTENT(OUT) :: parent_id ! Parent class property list
                                           ! identifier
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pget_class_parent_f
```

Name: H5Pget_copy_object

Signature:

herr_t `H5Pget_copy_object(` *hid_t* `ocp_plist_id`, *unsigned* *`copy_options`)

Purpose:

Retrieves the properties to be used when an object is copied.

Description:

`H5Pget_copy_object` retrieves the properties currently specified in the object copy property list `ocp_plist_id`, which will be invoked when a new copy is made of an existing object.

`copy_options` is a bit map indicating the flags, or properties, governing object copying that are set in the property list `ocp_plist_id`.

The available flags are described in `H5Pset_copy_object`.

Parameters:

hid_t `ocp_plist_id`	Object copy property list identifier
unsigned *`copy_options`	Copy option(s) set in the object copy property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_copy_object_f

```
SUBROUTINE h5pset_copy_object_f(ocp_plist_id, copy_options, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: ocp_plist_id
                                      ! Object copy property list identifier
  INTEGER, INTENT(IN) :: copy_options ! Copy option(s) to be set:
                                      !   H5O_COPY_SHALLOW_HIERARCHY_F
                                      !   H5O_COPY_EXPAND_SOFT_LINK_F
                                      !   H5O_COPY_EXPAND_EXT_LINK_F
                                      !   H5O_COPY_EXPAND_REFERENCE_F
                                      !   H5O_COPY_WITHOUT_ATTR_FLAG_F
  INTEGER, INTENT(OUT) :: hdferr      ! Error code
                                      ! 0 on success and -1 on failure
END SUBROUTINE h5pset_copy_object_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_create_intermediate_group

Signature:

herr_t `H5Pget_create_intermediate_group`(*hid_t* `lcpl_id`, *unsigned* `*crt_intermed_group`)

Purpose:

Determines whether property is set to enable creating missing intermediate groups.

Description:

`H5Pget_create_intermediate_group` determines whether the link creation property list `lcpl_id` is set to allow functions that create objects in groups different from the current working group to create intermediate groups that may be missing in the path of a new or moved object.

Functions that create objects in or move objects to a group other than the current working group make use of this property. `H5Gcreate_anon` and `H5Lmove` are examles of such functions.

If `crt_intermed_group` is true, missing intermediate groups will be created; if `crt_intermed_group` is false, missing intermediate groups will *not* be created.

Parameters:

hid_t `lcpl_id`	Link creation property list identifier
unsigned `*crt_intermed_group`	Flag specifying whether to create intermediate groups upon creation of an object

Returns:

Returns a non-negative valule if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_data_transform

Signature:

ssize_t `H5Pget_data_transform` (*hid_t* `plist_id`, *char* *`expression`, *size_t* `size`)

Purpose:

Retrieves a data transform expression.

Description:

`H5Pget_data_transform` retrieves the data transform expression previously set in the dataset transfer property list plist_id by `H5Pset_data_transform`.

`H5Pget_data_transform` can be used to both retrieve the transform expression and to query its size.

If `expression` is non-NULL, up to `size` bytes of the data transform expression are written to the buffer. If `expression` is NULL, `size` is ignored and the function does not write anything to the buffer. The function always returns the size of the data transform expression.

If `0` is returned for the size of the expression, no data transform expression exists for the property list.

If an error occurs, the buffer pointed to by `expression` is unchanged and the function returns a negative value.

Parameters:

hid_t `plist_id`	IN: Identifier of the property list or class
char *`expression`	OUT: Pointer to memory where the transform expression will be copied
size_t `size`	IN: Number of bytes of the transform expression to copy to

Returns:

Success: size of the transform expression.
Failure: a negative value.

Fortran90 Interface: h5pget_data_transform_f

```
SUBROUTINE h5pget_data_transform_f(plist_id, expression, hdferr, size)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id
                                     ! Identifier of the property list or class
  CHARACTER(LEN=*), INTENT(OUT) :: expression
                                     ! Buffer to hold transform expression
  INTEGER(SIZE_T), INTENT(OUT), OPTIONAL :: size
                                     ! Registered size for transform expression
  INTEGER, INTENT(OUT) :: hdferr     ! Error code
                                     ! 0 on success and -1 on failure
END SUBROUTINE h5pget_data_transform_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_driver

Signature:

hid_t `H5Pget_driver(` *hid_t* `plist_id )`

Purpose:

Returns low-lever driver identifier.

Description:

`H5Pget_driver` returns the identifier of the low-level file driver associated with the file access property list or data transfer property list `plist_id`.

Valid driver identifiers with the standard HDF5 library distribution include the following:

```
H5FD_CORE
H5FD_DIRECT
H5FD_FAMILY
H5FD_LOG
H5FD_MPIO
H5FD_MULTI
H5FD_SEC2
H5FD_STDIO
H5FD_WINDOWS  (Windows only)
```

If a user defines and registers custom drivers or if additional drivers are defined in an HDF5 distribution, this list will be longer.

The Windows driver, `H5FD_WINDOWS`, is available only on Windows systems.

The returned driver identifier is only valid as long as the file driver remains registered.

Parameters:

hid_t `plist_id`	IN: File access or data transfer property list identifier.

Returns:

Returns a valid low-level driver identifier if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_driver_f

```
SUBROUTINE h5pget_driver_f(prp_id, driver, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(OUT) :: driver       ! Low-level file driver identifier
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pget_driver_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pget_driver_info

Signature:

void *`H5Pget_driver_info`(*hid_t* `plist_id`)

Purpose:

Returns a pointer to file driver information.

Description:

`H5Pget_driver_info` returns a pointer to file driver-specific information for the low-level driver associated with the file access or data transfer property list `plist_id`.

The pointer returned by this function points to an “uncopied” struct. Driver-specific versions of that struct are defined for each low-level driver in the relevant source code file `H5FD*.c`. For example, the struct used for the `MULTI` driver is `H5FD_multi_fapl_t` defined in `H5FDmulti.c`.

If no driver-specific properties have been registered, `H5Pget_driver_info` returns `NULL`.

Note:

`H5Pget_driver_info` and `H5Pset_driver` are used only when creating a virtual file driver (VFD) in the virtual file layer (VFL). For further information, see “Virtual File Layer” and “List of VFL Functions” in the *HDF5 Technical Notes*.

Parameters:

hid_t `plist_id`

IN: File access or data transfer property list identifier.

Returns:

Returns a pointer to a struct containing low-level driver information. Otherwise returns `NULL`.

`NULL` is also returned if no driver-specific properties have been registered. No error is pushed on the stack in this case.

Non-C API(s):

None.

History:

Release	**C**
1.8.2	Function publicized in this release; previous releases described this function only in the virtual file driver documentation.

Name: H5Pget_dxpl_mpio
Signature:
herr_t `H5Pget_dxpl_mpio`(*hid_t* `dxpl_id`, *H5FD_mpio_xfer_t* *`xfer_mode`)
Purpose:
Returns the data transfer mode.
Description:
`H5Pget_dxpl_mpio` queries the data transfer mode currently set in the data transfer property list `dxpl_id`.

Upon return, `xfer_mode` contains the data transfer mode, if it is non-null.

`H5Pget_dxpl_mpio` is not a collective function.
Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier.
H5FD_mpio_xfer_t *`xfer_mode`	OUT: Data transfer mode.

Returns:
Returns a non-negative value if successful. Otherwise returns a negative value.
Fortran90 Interface: h5pget_dxpl_mpio_f
```
SUBROUTINE h5pget_dxpl_mpio_f(prp_id, data_xfer_mode, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id    ! Property list identifier
  INTEGER, INTENT(OUT) :: data_xfer_mode  ! Data transfer mode
                                          ! Possible values are:
                                          !    H5FD_MPIO_INDEPENDENT_F
                                          !    H5FD_MPIO_COLLECTIVE_F
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_dxpl_mpio_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pget_dxpl_multi

Signature:

herr_t `H5Pget_dxpl_multi`(*hid_t* `dxpl_id`, *const hid_t* *`memb_dxpl`)

Purpose:

Returns multi-file data transfer property list information.

Description:

`H5Pget_dxpl_multi` returns the data transfer property list information for the multi-file driver.

Parameters:

hid_t `dxpl_id`,	IN: Data transfer property list identifier.
const hid_t *`memb_dxpl`	OUT: Array of data access property lists.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pget_edc_check

Signature:

H5Z_EDC_t `H5Pget_edc_check`(*hid_t* `plist`)

Purpose:

Determines whether error-detection is enabled for dataset reads.

Description:

`H5Pget_edc_check` queries the dataset transfer property list `plist` to determine whether error detection is enabled for data read operations.

Parameters:

hid_t `plist`	IN: Dataset transfer property list identifier.

Returns:

Returns `H5Z_ENABLE_EDC` or `H5Z_DISABLE_EDC` if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_edc_check_f

```
SUBROUTINE h5pget_edc_check_f(prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Dataset transfer property list
                                        ! identifier
  INTEGER, INTENT(OUT)       :: flag    ! EDC flag; possible values
                                        !    H5Z_DISABLE_EDC_F
                                        !    H5Z_ENABLE_EDC_F
  INTEGER, INTENT(OUT)       :: hdferr  ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pget_edc_check_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Last modified: 8 May 2009

Name: H5Pget_elink_acc_flags

Signature:

herr_t `H5Pget_elink_acc_flags`(*hid_t* `lapl_id`, *unsigned* *`flags`)

Purpose:

Retrieves the external link traversal file access flag from the specified link access property list.

Description:

`H5Pget_elink_acc_flags` retrieves the file access flag used to open an external link target file from the specified link access property list.

The value returned, if it is not `H5F_ACC_DEFAULT` will override the default access flag, which is the access flag used to open the parent file.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier
unsigned *`flags`	OUT: File access flag for link traversal. Valid values include:

`H5F_ACC_RDWR`	Files opened through external links will be opened with write access.
`H5F_ACC_RDONLY`	Files opened through external links will be opened with read-only access.
`H5F_ACC_DEFAULT`	Files opened through external links will be opened with the same access flag as the parent file.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The following code retrieves the external link access flag settings on the link access property list `lapl_id` into a local variable:

```
unsigned acc_flags;
status = H5Pget_elink_acc_flags(lapl_id, &acc_flags);
```

See Also:

`H5Pset_elink_acc_flags`

History:

Release	**Change**
1.8.3	C function introduced in this release.

Last modified: 8 May 2009

Name: H5Pget_elink_cb

Signature:

herr_t `H5Pget_elink_cb`(*hid_t* `lapl_id`, *H5L_elink_traverse_t* **`func`, *void* **`op_data`)

Purpose:

Retrieves the external link traversal callback function from the specified link access property list.

Description:

`H5Pget_elink_cb` retrieves the user-defined external link traversal callback function defined in the specified link access property list.

The callback function may adjust the file access property list and file access flag to use when opening a file through an external link. The callback will be made immediately before opening the target file.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier.
H5L_elink_traverse_t **`func`	OUT: User-defined external link traversal callback function.
void **`op_data`	OUT: User-defined input data for the callback function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The following code retrieves the external link callback settings on the link access property list `lapl_id` into local variables:

```
H5L_elink_traverse_t *elink_callback_func;
void *elink_callback_udata;
status = H5Pget_elink_cb(lapl_id, &elink_callback_func, &elink_callback_udata);
```

See Also:

`H5Pset_elink_cb`,

`H5L_elink_traverse_t`

History:

Release	Change
1.8.3	C function introduced in this release.

Name: H5Pget_elink_fapl

Signature:

hid_t `H5Pget_elink_fapl`(*hid_t* `lapl_id`)

Purpose:

Retrieves the file access property list identifier associated with the link access property list.

Description:

`H5Pget_elink_fapl` retrieves the file access property list identifier that is set for the link access property list identifier, `lapl_id`. The library uses this file access property list identifier to open the target file for the external link access.

When no such identifier is set, this routine returns `H5P_DEFAULT`.

See also H5Pset_elink_fapl and H5Lcreate_external.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.2	Function introduced in this release.

Name: H5Pget_elink_prefix

Signature:

ssize_t `H5Pget_elink_prefix`(*hid_t* `lapl_id`, *char* *`prefix`, *size_t* `size`)

Purpose:

Retrieves prefix applied to external link paths.

Description:

`H5Pget_elink_prefix` retrieves the prefix applied to the path of any external links traversed.

When an external link is traversed, the prefix is retrieved from the link access property list `lapl_id`, returned in the user-allocated buffer pointed to by `prefix`, and prepended to the filename stored in the external link.

The size in bytes of the prefix, including the `NULL` terminator, is specified in `size`. If `size` is unknown, a preliminary `H5Pget_elink_prefix` call with the pointer `prefix` set to `NULL` will return the size of the prefix *without* the `NULL` terminator.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier
char *`prefix`	OUT: Prefix applied to external link paths
size_t `size`	IN: Size of prefix, including null terminator

Returns:

If successful, returns a non-negative value specifying the size in bytes of the prefix *without* the `NULL` terminator; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_est_link_info

Signature:

herr_t `H5Pget_est_link_info`(*hid_t* `gcpl_id`, *unsigned* `*est_num_entries`, *unsigned* `*est_name_len`)

Purpose:

Queries data required to estimate required local heap or object header size.

Description:

`H5Pget_est_link_info` queries a group creation property list, `gcpl_id`, for its "estimated number of links" and "estimated average name length" settings.

The estimated number of links anticipated to be inserted into a group created with this property list is returned in `est_num_entries`.

The estimated average length of the anticipated link names is returned in `est_name_len`.

The values for these two settings are multiplied to compute the initial local heap size (for old-style groups, if the local heap size hint is not set) or the initial object header size for (new-style compact groups; see "Group implementations in HDF5"). Accurately setting these parameters will help reduce wasted file space.

A value of `0` (zero) in `est_num_entries` will prevent a group from being created in the compact format.

See "Group implementations in HDF5" in the H5G API introduction for a discussion of the available types of HDF5 group structures.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
unsigned `*est_num_entries`	OUT: Estimated number of links to be inserted into group
unsigned `*est_name_len`	OUT: Estimated average length of link names

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_est_link_info_f

```
SUBROUTINE h5pget_est_link_info_f(gcpl_id,est_num_entries, est_name_len,hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id   ! Group creation property list id
  INTEGER, INTENT(OUT) :: est_num_entries ! Estimated number of links to be
                                          ! inserted into group
  INTEGER, INTENT(OUT) :: est_name_len    ! Estimated average length of link
                                          ! names
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_est_link_info_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_external

Signature:

herr_t `H5Pget_external`(*hid_t* `plist`, *unsigned* `idx`, *size_t* `name_size`, *char* `*name`, *off_t* `*offset`, *hsize_t* `*size`)

Purpose:

Returns information about an external file.

Description:

`H5Pget_external` returns information about an external file. The external file is specified by its index, `idx`, which is a number from zero to N-1, where N is the value returned by `H5Pget_external_count`. At most `name_size` characters are copied into the `name` array. If the external file name is longer than `name_size` with the null terminator, the return value is not null terminated (similar to `strncpy()`).

If `name_size` is zero or `name` is the null pointer, the external file name is not returned. If `offset` or `size` are null pointers then the corresponding information is not returned.

Parameters:

hid_t `plist`	IN: Identifier of a dataset creation property list.
unsigned `idx`	IN: External file index.
size_t `name_size`	IN: Maximum length of `name` array.
char `*name`	OUT: Name of the external file.
off_t `*offset`	OUT: Pointer to a location to return an offset value.
hsize_t `*size`	OUT: Pointer to a location to return the size of the external file data.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_external_f

```
SUBROUTINE h5pget_external_f(prp_id, idx, name_size, name, offset,bytes, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id     ! Property list identifier
  INTEGER, INTENT(IN) :: idx               ! External file index.
  INTEGER(SIZE_T), INTENT(IN) :: name_size ! Maximum length of name array
  CHARACTER(LEN=*), INTENT(OUT) :: name    ! Name of an external file
  INTEGER, INTENT(OUT) :: offset           ! Offset, in bytes, from the
                                           ! beginning of the file to the
                                           ! location in the file where
                                           ! the data starts.
  INTEGER(HSIZE_T), INTENT(OUT) :: bytes   ! Number of bytes reserved in
                                           ! the file for the data
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pget_external_f
```

History:

Release	**C**
1.6.4	`idx` parameter type changed to *unsigned*.

Name: H5Pget_external_count
Signature:
int `H5Pget_external_count`(*hid_t* `plist`)
Purpose:
Returns the number of external files for a dataset.
Description:
`H5Pget_external_count` returns the number of external files for the specified dataset.
Parameters:

hid_t `plist`	IN: Identifier of a dataset creation property list.

Returns:
Returns the number of external files if successful; otherwise returns a negative value.
Fortran90 Interface: h5pget_external_count_f

```
SUBROUTINE h5pget_external_count_f (prp_id, count, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(OUT) :: count        ! Number of external files for
                                       ! the specified dataset
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pget_external_count_f
```

Name: H5Pget_family_offset

Signature:

herr_t `H5Pget_family_offset` (*hid_t* `fapl_id`, *hsize_t* *`offset`)

Purpose:

Retrieves a data offset from the file access property list.

Description:

`H5Pget_family_offset` retrieves the value of `offset` from the file access property list `fapl_id` so that the user application can retrieve a file handle for low-level access to a particular member of a family of files. The file handle is retrieved with a separate call to `H5Fget_vfd_handle` (or, in special circumstances, to `H5FDget_vfd_handle`; see *Virtual File Layer* and *List of VFL Functions* in *HDF5 Technical Notes*).

The data offset returned in `offset` is the offset of the data in the HDF5 file that is stored on disk in the selected member file in a family of files.

Use of this function is only appropriate for an HDF5 file written as a family of files with the `FAMILY` file driver.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t *`offset`	OUT: Offset in bytes within the HDF5 file.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pget_fapl_core

Signature:

herr_t `H5Pget_fapl_core`(*hid_t* `fapl_id`, *size_t* `*increment`, *hbool_t* `*backing_store`)

Purpose:

Queries core file driver properties.

Description:

`H5Pget_fapl_core` queries the `H5FD_CORE` driver properties as set by `H5Pset_fapl_core`.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
size_t `*increment`	OUT: Size, in bytes, of memory increments.
hbool_t `*backing_store`	OUT: Boolean flag indicating whether to write the file contents to disk when the file is closed.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_fapl_core_f

```
SUBROUTINE h5pget_fapl_core_f(prp_id, increment, backing_store, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: prp_id     ! Property list identifier
  INTEGER(SIZE_T), INTENT(OUT) :: increment ! File block size in bytes
  LOGICAL, INTENT(OUT) :: backing_store     ! Flag to indicate that entire
                                            ! file contents are flushed to
                                            ! a file with the same name as
                                            ! this core file
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5pget_fapl_core_f
```

History:

Release	C	Fortran90
1.6.0		The `backing_store` parameter type changed from *INTEGER* to *LOGICAL* to better match the C API
1.4.0	Function introduced in this release.	

Name: H5Pget_fapl_direct

Signature:

herr_t `H5Pget_fapl_direct`(*hid_t* `fapl_id`, *size_t* `*alignment`, *size_t* `*block_size`, *size_t* `*cbuf_size`)

Purpose:

Retrieves direct I/O driver settings.

Description:

`H5Pget_fapl_direct` retrieves the required memory alignment (`alignment`), file system block size (`block_size`), and copy buffer size (`cbuf_size`) settings for the direct I/O driver, `H5FD_DIRECT`, from the file access property list `fapl_id`.

See `H5Pset_fapl_direct` for discussion of these values, requirements, and important considerations.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
size_t `*alignment`	OUT: Required memory alignment boundary
size_t `*block_size`	OUT: File system block size
size_t `*cbuf_size`	OUT: Copy buffer size

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE H5Pget_fapl_direct_f(fapl_id, alignment, block_size, cbuf_size, &
                                hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: fapl_id ! File access property list identifier
  INTEGER(SIZE_T), INTENT(OUT) :: alignment
                                        ! Required memory alignment boundary!
  INTEGER(SIZE_T), INTENT(OUT) :: block_size
                                        ! File system block size
  INTEGER(SIZE_T), INTENT(OUT) :: cbuf_size
                                        ! Copy buffer size
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE H5Pget_fapl_direct_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_fapl_family

Signature:

herr_t `H5Pget_fapl_family` (*hid_t* `fapl_id`, *hsize_t* *`memb_size`, *hid_t* *`memb_fapl_id`)

Purpose:

Returns file access property list information.

Description:

`H5Pget_fapl_family` returns file access property list for use with the family driver. This information is returned through the output parameters.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t *`memb_size`	OUT: Size in bytes of each file member.
hid_t *`memb_fapl_id`	OUT: Identifier of file access property list for each family member.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_fapl_family_f

```
SUBROUTINE h5pget_fapl_family_f(prp_id, imemb_size, memb_plist, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)    :: prp_id     ! Property list identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: memb_size ! Logical size, in bytes,
                                              ! of each family member
  INTEGER(HID_T), INTENT(OUT) :: memb_plist  ! Identifier of the file
                                              ! access property list to be
                                              ! used for each family member
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure
END SUBROUTINE h5pget_fapl_family_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Last modified: 15 May 2009

Name: H5Pget_fapl_mpio

Signature:

herr_t `H5Pget_fapl_mpio`(*hid_t* `fapl_id`, *MPI_Comm* *`comm`, *MPI_Info* *`info`)

Purpose:

Returns MPI communicator information.

Description:

If the file access property list is set to the `H5FD_MPIO` driver, `H5Pget_fapl_mpio` returns duplicates of the stored MPI communicator and Info object through the `comm` and `info` pointers, if those values are non-null.

Since the MPI communicator and Info object are duplicates of the stored information, future modifications to the access property list will not affect them. It is the responsibility of the application to free these objects.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
MPI_Comm *`comm`	OUT: MPI-2 communicator
MPI_Info *`info`	OUT: MPI-2 Info object

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_fapl_mpio_f

```
SUBROUTINE h5pget_fapl_mpio_f(prp_id, comm, info, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(OUT) :: comm         ! Buffer to return communicator
  INTEGER, INTENT(IN) :: info          ! Buffer to return info object as
                                       ! defined in MPI_FILE_OPEN of MPI-2
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pget_fapl_mpio_f
```

History:

Release	Change
1.4.5	Handling of the MPI Communicator and Info object changed at this release. A duplicate of each of these objects is now returned instead of pointers to each object.
1.4.0	C function introduced in this release.

Name: H5Pget_fapl_mpiposix

Signature:

herr_t `H5Pget_fapl_mpiposix`(*hid_t* `fapl_id`, *MPI_Comm* *`comm`, *hbool_t* *`use_gpfs_hints`)

Purpose:

Returns MPI communicator information.

Description:

If the file access property list is set to the `H5FD_MPIO` driver, `H5Pget_fapl_mpiposix` returns the MPI communicator through the `comm` pointer, if those values are non-null.

`comm` is not copied, so it is valid only until the file access property list is either modified or closed.

`use_gpfs_hints` specifies whether to attempt to use GPFS hints when accessing this file. A value of `TRUE` (or `1`) indicates that the hints are being used, where possible. A value of `FALSE` (or `0`) indicates that the hints are not being used.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
MPI_Comm *`comm`	OUT: MPI-2 communicator.
hbool_t *`use_gpfs_hints`	OUT: Use of GPFS hints.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_fapl_mpiposix_f

```
SUBROUTINE h5pget_fapl_mpiposix_f(prp_id, comm, use_gpfs, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(OUT) :: comm          ! Buffer to return communicator
  LOGICAL, INTENT(OUT) :: use_gpfs
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5pget_fapl_mpiposix_f
```

History:

Release	**C**	**Fortran90**
1.6.1		Fortran subroutine introduced in this release.
1.6.0	`use_gpfs_hints` parameter added.	
1.6.0	Function introduced in this release.	

Name: H5Pget_fapl_multi

Signature:

herr_t `H5Pget_fapl_multi`(*hid_t* `fapl_id`, *const H5FD_mem_t* *`memb_map`, *const hid_t* *`memb_fapl`, *const char* **`memb_name`, *const haddr_t* *`memb_addr`, *hbool_t* *`relax`)

Purpose:

Returns information about the multi-file access property list.

Description:

`H5Pget_fapl_multi` returns information about the multi-file access property list.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
const H5FD_mem_t *`memb_map`	OUT: Maps memory usage types to other memory usage types.
const hid_t *`memb_fapl`	OUT: Property list for each memory usage type.
const char **`memb_name`	OUT: Name generator for names of member files.
const haddr_t *`memb_addr`	OUT: The offsets within the virtual address space, from 0 (zero) to `HADDR_MAX`, at which each type of data storage begins.
hbool_t *`relax`	OUT: Allows read-only access to incomplete file sets when `TRUE`.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_fapl_multi_f

```
SUBROUTINE h5pget_fapl_multi_f(prp_id, memb_map, memb_fapl, memb_name,
                               memb_addr, relax, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T),INTENT(IN)   :: prp_id    ! Property list identifier

  INTEGER,DIMENSION(0:H5FD_MEM_NTYPES_F-1),INTENT(OUT)          :: memb_map
  INTEGER(HID_T),DIMENSION(0:H5FD_MEM_NTYPES_F-1),INTENT(OUT)   :: memb_fapl
  CHARACTER(LEN=*),DIMENSION(0:H5FD_MEM_NTYPES_F-1),INTENT(OUT) :: memb_name
  REAL, DIMENSION(0:H5FD_MEM_NTYPES_F-1), INTENT(OUT)           :: memb_addr
             ! Numbers in the interval [0,1) (e.g. 0.0 0.1 0.5 0.2 0.3 0.4)
             ! real address in the file will be calculated as X*HADDR_MAX

  LOGICAL, INTENT(OUT) :: relax
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pget_fapl_multi_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pget_fclose_degree

Signature:

herr_t `H5Pget_fclose_degree`(*hid_t* `fapl_id`, *H5F_close_degree_t* *`fc_degree`)

Purpose:

Returns the file close degree.

Description:

`H5Pget_fclose_degree` returns the current setting of the file close degree property `fc_degree` in the file access property list `fapl_id`.

The value of `fc_degree` determines how aggressively `H5Fclose` deals with objects within a file that remain open when `H5Fclose` is called to close that file. `fc_degree` can have any one of four valid values as described in `H5Pset_fclose_degree`.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
H5F_close_degree_t *`fc_degree`	OUT: Pointer to a location to which to return the file close degree property, the value of `fc_degree`.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_fclose_degree_f

```
SUBROUTINE h5pget_fclose_degree_f(fapl_id, degree, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: fapl_id ! File access property list identifier
  INTEGER, INTENT(OUT) :: degree        ! Info about file close behavior
                                        ! Possible values:
                                        !    H5F_CLOSE_DEFAULT_F
                                        !    H5F_CLOSE_WEAK_F
                                        !    H5F_CLOSE_SEMI_F
                                        !    H5F_CLOSE_STRONG_F
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pget_fclose_degree_f
```

History:

Release	C
1.6.0	Function introduced in this release.

Name: H5Pget_fill_time

Signature:

herr_t `H5Pget_fill_time`(*hid_t* `plist_id`, *H5D_fill_time_t* *`fill_time`)

Purpose:

Retrieves the time when fill value are written to a dataset.

Description:

`H5Pget_fill_time` examines the dataset creation property list `plist_id` to determine when fill values are to be written to a dataset.

Valid values returned in `fill_time` are as follows:

`H5D_FILL_TIME_IFSET`	Fill values are written to the dataset when storage space is allocated only if there is a user-defined fill value, i.e., one set with H5Pset_fill_value. (Default)
`H5D_FILL_TIME_ALLOC`	Fill values are written to the dataset when storage space is allocated.
`H5D_FILL_TIME_NEVER`	Fill values are never written to the dataset.

Note:

`H5Pget_fill_time` is designed to work in coordination with the dataset fill value and dataset storage allocation time properties, retrieved with the functions `H5Pget_fill_value` and `H5Pget_alloc_time`.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
H5D_fill_time_t *`fill_time`	OUT: Setting for the timing of writing fill values to the dataset.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_fill_time_f

```
SUBROUTINE h5pget_fill_time_f(plist_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! Dataset creation property
                                         ! list identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: flag  ! Fill time flag
                                         ! Possible values are:
                                         !    H5D_FILL_TIME_ERROR_F
                                         !    H5D_FILL_TIME_ALLOC_F
                                         !    H5D_FILL_TIME_NEVER_F
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pget_fill_time_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pget_fill_value

Signature:

herr_t `H5Pget_fill_value`(*hid_t* `plist_id`, *hid_t* `type_id`, *void* *`value`)

Purpose:

Retrieves a dataset fill value.

Description:

`H5Pget_fill_value` returns the dataset fill value defined in the dataset creation property list `plist_id`.

The fill value is returned through the `value` pointer and will be converted to the datatype specified by `type_id`. This datatype may differ from the fill value datatype in the property list, but the HDF5 library must be able to convert between the two datatypes.

If the fill value is undefined, i.e., set to `NULL` in the property list, `H5Pget_fill_value` will return an error. `H5Pfill_value_defined` should be used to check for this condition before `H5Pget_fill_value` is called.

Memory must be allocated by the calling application.

Note:

`H5Pget_fill_value` is designed to coordinate with the dataset storage allocation time and fill value write time properties, which can be retrieved with the functions `H5Pget_alloc_time` and `H5Pget_fill_time`, respectively.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
hid_t `type_id`,	IN: Datatype identifier for the value passed via `value`.
void *`value`	OUT: Pointer to buffer to contain the returned fill value.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_fill_value_f

```
SUBROUTINE h5pget_fill_value_f(prp_id, type_id, fillvalue, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier of fill
                                        ! value datatype (in memory)
  TYPE(VOID), INTENT(IN) :: fillvalue   ! Fillvalue
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure

END SUBROUTINE h5pget_fill_value_f
```

Name: H5Pget_filter

Signatures:

H5Z_filter_t `H5Pget_filter`(*hid_t* `plist`, *unsigned int* `idx`, *unsigned int* *`flags`, *size_t* *`cd_nelmts`, *unsigned int* *`cd_values`, *size_t* `namelen`, *char* `name[]`) [1]

H5Z_filter_t `H5Pget_filter`(*hid_t* `plist_id`, *unsigned* `idx`, *unsigned int* *`flags`, *size_t* *`cd_nelmts`, *unsigned* `cd_values[]`, *size_t* `namelen`, *char* `name[]`, *unsigned* *`filter_config`) [2]

Purpose:

Returns information about a filter in a pipeline.

Description:

`H5Pget_filter` is a macro that is mapped to either `H5Pget_filter1` or `H5Pget_filter2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

- ◊ The `H5Pget_filter` macro will be mapped to `H5Pget_filter1` and will use the `H5Pget_filter1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.
- ◊ The `H5Pget_filter` macro mapped to `H5Pget_filter2` and will use the `H5Pget_filter2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in “API Compatibility Macros in HDF5” we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Pget_filter` is mapped to the most recent version of the function, currently `H5Pget_filter2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Pget_filter` will be mapped to `H5Pget_filter1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Pget_filter` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Pget_filter2`
Enable deprecated symbols	`H5Pget_filter2`
Disable deprecated symbols	`H5Pget_filter2`
Emulate Release 1.6 interface	`H5Pget_filter1`

Function-level macros

`H5Pget_filter_vers = 2    H5Pget_filter2`

`H5Pget_filter_vers = 1    H5Pget_filter1`

Interface history: Signature [1] above is the original `H5Pget_filter` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Pget_filter1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Pget_filter2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface: h5pget_filter_f

```
SUBROUTINE h5pget_filter_f(prp_id, filter_number, flags, cd_nelmts,
                           cd_values, namelen, name, filter_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id    ! Property list identifier
  INTEGER, INTENT(IN) :: filter_number    ! Sequence number within the filter
                                          ! pipeline of the filter for which
                                          ! information is sought
  INTEGER, DIMENSION(*), INTENT(OUT) :: cd_values
                                          ! Auxiliary data for the filter
  INTEGER, INTENT(OUT) :: flags           ! Bit vector specifying certain
                                          ! general properties of the filter
  INTEGER(SIZE_T), INTENT(INOUT) :: cd_nelmts
                                          ! Number of elements in cd_values
  INTEGER(SIZE_T), INTENT(IN) :: namelen  ! Anticipated number of characters
                                          ! in name
  CHARACTER(LEN=*), INTENT(OUT) :: name   ! Name of the filter
  INTEGER, INTENT(OUT) :: filter_id       ! Filter identification number
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_filter_f
```

History:

Release	**C**
1.8.0	The function `H5Pget_filter` renamed to `H5Pget_filter1` and deprecated in this release. The macro `H5Pget_filter` and the function `H5Pget_filter2` introduced in this release.

Name: H5Pget_filter1

Signature:

H5Z_filter_t `H5Pget_filter1`(*hid_t* `plist`, *unsigned int* `idx`, *unsigned int* *`flags`, *size_t* *`cd_nelmts`, *unsigned int* *`cd_values`, *size_t* `namelen`, *char* `name[]`)

Purpose:

Returns information about a filter in a pipeline.

Notice:

This function is renamed from `H5Pget_filter` and deprecated in favor of the function `H5Pget_filter2` or the new macro `H5Pget_filter`.

Description:

`H5Pget_filter1` returns information about a filter, specified by its filter number, in a filter pipeline, specified by the property list with which it is associated.

If `plist` is a dataset creation property list, the pipeline is a permanent filter pipeline; if `plist` is a dataset transfer property list, the pipeline is a transient filter pipeline.

`idx` is a value between zero and *N*-1, as described in `H5Pget_nfilters`. The function will return a negative value if the filter number is out of range.

The structure of the `flags` argument is discussed in `H5Pset_filter`.

On input, `cd_nelmts` indicates the number of entries in the `cd_values` array, as allocated by the caller; on return,`cd_nelmts` contains the number of values defined by the filter.

If `name` is a pointer to an array of at least `namelen` bytes, the filter name will be copied into that array. The name will be null terminated if `namelen` is large enough. The filter name returned will be the name appearing in the file, the name registered for the filter, or an empty string.

Note:

This function currently supports only the permanent filter pipeline; `plist` must be a dataset creation property list.

Parameters:

hid_t `plist`	IN: Property list identifier.
int `idx`	IN: Sequence number within the filter pipeline of the filter for which information is sought.
unsigned int *`flags`	OUT: Bit vector specifying certain general properties of the filter.
size_t *`cd_nelmts`	IN/OUT: Number of elements in `cd_values`.
unsigned int *`cd_values`	OUT: Auxiliary data for the filter.
size_t `namelen`	IN: Anticipated number of characters in `name`.
char `name[]`	OUT: Name of the filter.

Returns:

Returns the filter identifier if successful:

`H5Z_FILTER_DEFLATE`	Data compression filter, employing the gzip algorithm
`H5Z_FILTER_SHUFFLE`	Data shuffling filter
`H5Z_FILTER_FLETCHER32`	Error detection filter, employing the Fletcher32 checksum algorithm
`H5Z_FILTER_SZIP`	Data compression filter, employing the SZIP algorithm
`H5Z_FILTER_NBIT`	Data compression filter, employing the N-bit algorithm
`H5Z_FILTER_SCALEOFFSET`	Data compression filter, employing the scale-offset algorithm

Otherwise returns a negative value.

Fortran90 Interface: h5pget_filter_f

```
SUBROUTINE h5pget_filter_f(prp_id, filter_number, flags, cd_nelmts,
                           cd_values, namelen, name, filter_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id    ! Property list identifier
  INTEGER, INTENT(IN) :: filter_number    ! Sequence number within the filter
                                          ! pipeline of the filter for which
                                          ! information is sought
  INTEGER, DIMENSION(*), INTENT(OUT) :: cd_values
                                          ! Auxiliary data for the filter
  INTEGER, INTENT(OUT) :: flags           ! Bit vector specifying certain
                                          ! general properties of the filter
  INTEGER(SIZE_T), INTENT(INOUT) :: cd_nelmts
                                          ! Number of elements in cd_values
  INTEGER(SIZE_T), INTENT(IN) :: namelen  ! Anticipated number of characters
                                          ! in name
  CHARACTER(LEN=*), INTENT(OUT) :: name   ! Name of the filter
  INTEGER, INTENT(OUT) :: filter_id       ! Filter identification number
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_filter_f
```

History:

Release	**C**
1.6.4	`filter` parameter type changed to *unsigned*.
1.8.0	N-bit and scale-offset filters added.
1.8.0	Function `H5Pget_filter` renamed to `H5Pget_filter1` and deprecated in this release.

Name: H5Pget_filter2

Signature:

H5Z_filter_t `H5Pget_filter2`(*hid_t* `plist_id`, *unsigned* `idx`, *unsigned int* `*flags`, *size_t* `*cd_nelmts`, *unsigned* `cd_values[]`, *size_t* `namelen`, *char* `name[]`, *unsigned* `*filter_config`)

Purpose:

Returns information about a filter in a pipeline.

Description:

`H5Pget_filter2` returns information about a filter, specified by its filter number, in a filter pipeline, specified by the property list with which it is associated.

If `plist` is a dataset creation property list, the pipeline is a permanent filter pipeline; if `plist` is a dataset transfer property list, the pipeline is a transient filter pipeline.

`idx` is a value between zero and *N*-1, as described in `H5Pget_nfilters`. The function will return a negative value if the filter number is out of range.

The structure of the `flags` argument is discussed in `H5Pset_filter`.

On input, `cd_nelmts` indicates the number of entries in the `cd_values` array, as allocated by the caller; on return,`cd_nelmts` contains the number of values defined by the filter.

If `name` is a pointer to an array of at least `namelen` bytes, the filter name will be copied into that array. The name will be null terminated if `namelen` is large enough. The filter name returned will be the name appearing in the file, the name registered for the filter, or an empty string.

`filter_config` is the bit field described in `H5Zget_filter_info`.

Note:

This function currently supports only the permanent filter pipeline; `plist` must be a dataset creation property list.

Parameters:

hid_t `plist`	IN: Property list identifier.
int `idx`	IN: Sequence number within the filter pipeline of the filter for which information is sought.
unsigned int `*flags`	OUT: Bit vector specifying certain general properties of the filter.
size_t `*cd_nelmts`	IN/OUT: Number of elements in `cd_values`.
unsigned int `*cd_values`	OUT: Auxiliary data for the filter.
size_t `namelen`	IN: Anticipated number of characters in `name`.
char `name[]`	OUT: Name of the filter.
unsigned int `*filter_config`	OUT: Bit field, as described in `H5Zget_filter_info`.

Returns:

Returns the filter identifier if successful:

`H5Z_FILTER_DEFLATE`	Data compression filter, employing the gzip algorithm
`H5Z_FILTER_SHUFFLE`	Data shuffling filter
`H5Z_FILTER_FLETCHER32`	Error detection filter, employing the Fletcher32 checksum algorithm
`H5Z_FILTER_SZIP`	Data compression filter, employing the SZIP algorithm
`H5Z_FILTER_NBIT`	Data compression filter, employing the N-bit algorithm
`H5Z_FILTER_SCALEOFFSET`	Data compression filter, employing the scale-offset algorithm

Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_filter_by_id

Signatures:

herr_t `H5Pget_filter_by_id(` *hid_t* `plist_id`, *H5Z_filter_t* `filter_id`, *unsigned int* `*flags`, *size_t* `*cd_nelmts`, *unsigned int* `cd_values[]`, *size_t* `namelen`, *char* `name[]`) [1]

herr_t `H5Pget_filter_by_id(` *hid_t* `plist_id`, *H5Z_filter_t* `filter_id`, *unsigned int* `*flags`, *size_t* `*cd_nelmts`, *unsigned int* `cd_values[]`, *size_t* `namelen`, *char* `name[]`, *unsigned int* `*filter_config`) [2]

Purpose:

Returns information about the specified filter.

Description:

`H5Pget_filter_by_id` is a macro that is mapped to either `H5Pget_filter_by_id1` or `H5Pget_filter_by_id2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

- ◊ The `H5Pget_filter_by_id` macro will be mapped to `H5Pget_filter_by_id1` and will use the `H5Pget_filter_by_id1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.
- ◊ The `H5Pget_filter_by_id` macro mapped to `H5Pget_filter_by_id2` and will use the `H5Pget_filter_by_id2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in “API Compatibility Macros in HDF5” we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Pget_filter_by_id` is mapped to the most recent version of the function, currently `H5Pget_filter_by_id2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Pget_filter_by_id` will be mapped to `H5Pget_filter_by_id1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Pget_filter_by_id` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Pget_filter_by_id2`
Enable deprecated symbols	`H5Pget_filter_by_id2`
Disable deprecated symbols	`H5Pget_filter_by_id2`
Emulate Release 1.6 interface	`H5Pget_filter_by_id1`

Function-level macros

`H5Pget_filter_by_id_vers = 2`	`H5Pget_filter_by_id2`
`H5Pget_filter_by_id_vers = 1`	`H5Pget_filter_by_id1`

Interface history: Signature [1] above is the original `H5Pget_filter_by_id` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Pget_filter_by_id1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Pget_filter_by_id2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface: h5pget_filter_by_id_f

```
SUBROUTINE h5pget_filter_by_id_f(prp_id, filter_id, flags, cd_nelmts,
                                 cd_values, namelen, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id      ! Property list identifier
  INTEGER, INTENT(IN)        :: filter_id   ! Filter identifier
  INTEGER(SIZE_T), INTENT(INOUT)     :: cd_nelmts
                                            ! Number of elements in cd_values
  INTEGER, DIMENSION(*), INTENT(OUT) :: cd_values
                                            ! Auxiliary data for the filter
  INTEGER, INTENT(OUT)          :: flags    ! Bit vector specifying certain
                                            ! general properties of the filter
  INTEGER(SIZE_T), INTENT(IN)   :: namelen  ! Anticipated number of characters
                                            ! in name
  CHARACTER(LEN=*), INTENT(OUT) :: name     ! Name of the filter
  INTEGER, INTENT(OUT)          :: hdferr   ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5pget_filter_by_id_f
```

History:

Release	**C**
1.8.0	The function `H5Pget_filter_by_id` renamed to `H5Pget_filter_by_id1` and deprecated in this release. The macro `H5Pget_filter_by_id` and the function `H5Pget_filter_by_id2` introduced in this release.

Name: H5Pget_filter_by_id1

Signature:

herr_t `H5Pget_filter_by_id1`(*hid_t* `plist_id`, *H5Z_filter_t* `filter_id`, *unsigned int* `*flags`, *size_t* `*cd_nelmts`, *unsigned int* `cd_values[]`, *size_t* `namelen`, *char* `name[]`)

Purpose:

Returns information about the specified filter.

Notice:

This function is renamed from `H5Pget_filter_by_id` and deprecated in favor of the function `H5Pget_filter_by_id2` or the new macro `H5Pget_filter_by_id`.

Description:

`H5Pget_filter_by_id1` returns information about the filter specified in `filter_id`, a filter identifier.

`plist_id` must identify a dataset creation property list and `filter_id` will be in a permanent filter pipeline.

The `filter_id` and `flags` parameters are used in the same manner as described in the discussion of `H5Pset_filter`.

Aside from the fact that they are used for output, the parameters `cd_nelmts` and `cd_values[]` are used in the same manner as described in the discussion of `H5Pset_filter`. On input, the `cd_nelmts` parameter indicates the number of entries in the `cd_values[]` array allocated by the calling program; on exit it contains the number of values defined by the filter.

On input, the `name_len` parameter indicates the number of characters allocated for the filter name by the calling program in the array `name[]`. On exit it contains the length in characters of name of the filter. On exit `name[]` contains the name of the filter with one character of the name in each element of the array.

If the filter specified in `filter_id` is not set for the property list, an error will be returned and `H5Pget_filter_by_id1` will fail.

Parameters:

hid_t `plist_id`	IN: Property list identifier.
H5Z_filter_t `filter_id`	IN: Filter identifier.
unsigned int `*flags`	OUT: Bit vector specifying certain general properties of the filter.
size_t `cd_nelmts`	IN/OUT: Number of elements in `cd_values`.
unsigned int `*cd_values`	OUT: Auxiliary data for the filter.
size_t `namelen`	IN/OUT: Length of filter name and number of elements in `name[]`.
char `name[]`	OUT: Name of filter.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_filter_by_id_f

See the `H5Pget_filter_by_id` macro description.

History:

Release	**C**
1.6.0	Function introduced in this release.
1.8.0	Function `H5Tget_filter_by_id` renamed to `H5Tget_filter_by_id1` and deprecated in this release.

Name: H5Pget_filter_by_id2

Signature:

herr_t `H5Pget_filter_by_id2`(*hid_t* `plist_id`, *H5Z_filter_t* `filter_id`, *unsigned int* *`flags`, *size_t* *`cd_nelmts`, *unsigned int* `cd_values[]`, *size_t* `namelen`, *char* `name[]`, *unsigned int* *`filter_config`)

Purpose:

Returns information about the specified filter.

Description:

`H5Pget_filter_by_id2` returns information about the filter specified in `filter_id`, a filter identifier.

`plist_id` must identify a dataset creation property list and `filter_id` will be in a permanent filter pipeline.

The `filter_id` and `flags` parameters are used in the same manner as described in the discussion of `H5Pset_filter`.

Aside from the fact that they are used for output, the parameters `cd_nelmts` and `cd_values[]` are used in the same manner as described in the discussion of `H5Pset_filter`. On input, the `cd_nelmts` parameter indicates the number of entries in the `cd_values[]` array allocated by the calling program; on exit it contains the number of values defined by the filter.

On input, the `name_len` parameter indicates the number of characters allocated for the filter name by the calling program in the array `name[]`. On exit it contains the length in characters of name of the filter. On exit `name[]` contains the name of the filter with one character of the name in each element of the array.

`filter_config` is the bit field described in `H5Zget_filter_info`.

If the filter specified in `filter_id` is not set for the property list, an error will be returned and `H5Pget_filter_by_id2` will fail.

Parameters:

hid_t `plist_id`	IN: Property list identifier.
H5Z_filter_t `filter_id`	IN: Filter identifier.
unsigned int *`flags`	OUT: Bit vector specifying certain general properties of the filter.
size_t `cd_nelmts`	IN/OUT: Number of elements in `cd_values`.
unsigned int *`cd_values`	OUT: Auxiliary data for the filter.
size_t `namelen`	IN/OUT: Length of filter name and number of elements in `name[]`.
char `name[]`	OUT: Name of filter.
unsigned int *`filter_config`	OUT: Bit field, as described in `H5Zget_filter_info`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_filter_by_id_f

See the `H5Pget_filter_by_id` macro description.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_gc_references

Signature:

herr_t `H5Pget_gc_references`(*hid_t* `plist`, *unsigned* `*gc_ref`)

Purpose:

Returns garbage collecting references setting.

Description:

`H5Pget_gc_references` returns the current setting for the garbage collection references property from the specified file access property list. The garbage collection references property is set by `H5Pset_gc_references`.

Parameters:

hid_t `plist`	IN: File access property list identifier.
unsigned `gc_ref`	OUT: Flag returning the state of reference garbage collection. A returned value of `1` indicates that garbage collection is on while `0` indicates that garbage collection is off.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_gc_references_f

```
SUBROUTINE h5pget_gc_references_f (prp_id, gc_reference, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(OUT) :: gc_reference ! The flag for garbage collecting
                                       ! references for the file
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pget_gc_references_f
```

Name: H5Pget_hyper_vector_size

Signature:

herr_t `H5Pget_hyper_vector_size`(*hid_t* `dxpl_id`, *size_t* *`vector_size`)

Purpose:

Retrieves number of I/O vectors to be read/written in hyperslab I/O.

Description:

`H5Pset_hyper_vector_size` retrieves the number of I/O vectors to be accumulated in memory before being issued to the lower levels of the HDF5 library for reading or writing the actual data.

The number of I/O vectors set in the dataset transfer property list `dxpl_id` is returned in `vector_size`. Unless the default value is in use, `vector_size` was previously set with a call to H5Pset_hyper_vector_size.

Parameters:

hid_t `dxpl_id`	IN: Dataset transfer property list identifier.
size_t *`vector_size`	OUT: Number of I/O vectors to accumulate in memory for I/O operations.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_hyper_vector_size_f

```
SUBROUTINE h5pget_hyper_vector_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! Dataset transfer property list
                                         ! identifier
  INTEGER(SIZE_T), INTENT(OUT) :: size   ! Vector size
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pget_hyper_vector_size_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pget_istore_k

Signature:

herr_t `H5Pget_istore_k`(*hid_t* `plist`, *unsigned* * `ik`)

Purpose:

Queries the 1/2 rank of an indexed storage B-tree.

Description:

`H5Pget_istore_k` queries the 1/2 rank of an indexed storage B-tree. The argument `ik` may be the null pointer (NULL). This function is only valid for file creation property lists.

See H5Pset_istore_k for details.

Parameters:

hid_t `plist`	IN: Identifier of property list to query.
unsigned * `ik`	OUT: Pointer to location to return the chunked storage B-tree 1/2 rank.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_istore_k_f

```
SUBROUTINE h5pget_istore_k_f(prp_id, ik, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(OUT) :: ik           ! 1/2 rank of chunked storage B-tree
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pget_istore_k_f
```

History:

Release	**C**
1.6.4	`ik` parameter type changed to *unsigned*.

Name: H5Pget_layout

Signature:

H5D_layout_t `H5Pget_layout`(*hid_t* `plist`)

Purpose:

Returns the layout of the raw data for a dataset.

Description:

`H5Pget_layout` returns the layout of the raw data for a dataset. This function is only valid for dataset creation property lists.

Note that a compact storage layout may affect writing data to the dataset with parallel applications. See note in H5Dwrite documentation for details.

Parameters:

hid_t `plist`	IN: Identifier for property list to query.

Returns:

Returns the layout type (a non-negative value) of a dataset creation property list if successful. Valid return values are:

H5D_COMPACT
: Raw data is stored in the object header in the file.

H5D_CONTIGUOUS
: Raw data is stored separately from the object header in one contiguous chunk in the file.

H5D_CHUNKED
: Raw data is stored separately from the object header in chunks in separate locations in the file.

Otherwise, returns a negative value indicating failure.

Fortran90 Interface: h5pget_layout_f

```
SUBROUTINE h5pget_layout_f (prp_id, layout, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(OUT) :: layout       ! Type of storage layout for raw data
                                       ! possible values are:
                                       !    H5D_COMPACT_F
                                       !    H5D_CONTIGUOUS_F
                                       !    H5D_CHUNKED_F
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pget_layout_f
```

Name: H5Pget_libver_bounds

Signature:

herr_t `H5Pget_libver_bounds(` *hid_t* `fapl_id`, *H5F_libver_t* `*low`, *H5F_libver_t* `*high`)

Purpose:

Retrieves library version bounds settings that indirectly control the format versions used when creating objects.

Description:

`H5Pget_libver_bounds` retrieves the lower and upper bounds on the HDF5 Library versions that indirectly determine the object formats versions used when creating objects in the file.

This property is retrieved from the file access property list specified by `fapl_id`.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
H5F_libver_t `*low`	OUT: The earliest version of the library that will be used for writing objects. The library version indirectly specifies the earliest object format version that can be used when creating objects in the file. Currently, `H5F_LIBVER_EARLIEST` and `H5F_LIBVER_LATEST` are the only valid values for `low`.
H5F_libver_t `*high`	OUT: The latest version of the library that will be used for writing objects. The library version indirectly specifies the latest object format version that can be used when creating objects in the file. Currently, `H5F_LIBVER_LATEST` is the only valid value for `high`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_link_creation_order
Signature:

herr_t `H5Pget_link_creation_order(` *hid_t* `gcpl_id`, *unsigned* `*crt_order_flags`)

Purpose:

Queries whether link creation order is tracked and/or indexed in a group.

Description:

`H5Pget_link_creation_order` queries the group creation property list, `gcpl_id`, and returns a flag indicating whether link creation order is tracked and/or indexed in a group.

See `H5Pset_link_creation_order` for a list of valid creation order flags, as passed in `crt_order_flags`, and their meanings.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
unsigned `*crt_order_flags`	OUT: Creation order flag(s)

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_link_creation_order_f

```
SUBROUTINE h5pget_link_creation_order_f(gcpl_id, crt_order_flags, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id   ! Group creation property list id
  INTEGER, INTENT(OUT) :: crt_order_flags ! Creation order flag(s)
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_link_creation_order_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_link_phase_change
Signature:

herr_t `H5Pget_link_phase_change`(*hid_t* `gcpl_id`, *unsigned* *`max_compact`, *unsigned* *`min_dense`)

Purpose:

Queries the settings for conversion between compact and dense groups.

Description:

`H5Pget_link_phase_change` queries the maximum number of entries for a *compact group* and the minimum number links to require before converting a group to a *dense* form.

In the compact format, links are stored as messages in the group's header. In the dense format, links are stored in a fractal heap and indexed with a version 2 B-tree.

`max_compact` is the maximum number of links to store as header messages in the group header before converting the group to the dense format. Groups that are in the compact format and exceed this number of links are automatically converted to the dense format.

`min_dense` is the minimum number of links to store in the dense format. Groups which are in dense format and in which the number of links falls below this number are automatically converted back to the compact format.

In the compact format, links are stored as messages in the group's header. In the dense format, links are stored in a fractal heap and indexed with a version 2 B-tree.

See `H5Pset_link_phase_change` for a discussion of traditional, compact, and dense group storage.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
unsigned *`max_compact`	OUT: Maximum number of links for compact storage
unsigned *`min_dense`	OUT: Minimum number of links for dense storage

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE h5pset_link_phase_change_f(gcpl_id, max_compact, min_dense, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id
                                  ! Group creation property list identifier
  INTEGER, INTENT(IN) :: max_compact
                                  ! Maximum number of attributes to be stored
                                  ! in compact storage
  INTEGER, INTENT(IN) :: min_dense
                                  ! Minimum number of attributes to be stored
                                  ! in dense storage
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
                                  ! 0 on success and -1 on failure
END SUBROUTINE h5pset_link_phase_change_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_local_heap_size_hint

Signature:

herr_t `H5Pget_local_heap_size_hint`(*hid_t* `gcpl_id`, *size_t* *`size_hint`)

Purpose:

Retrieves the anticipated size of the local heap for original-style groups.

Description:

`H5Pget_local_heap_size_hint` queries the group creation property list, `gcpl_id`, for the anticipated size of the local heap, `size_hint`, for original-style groups, i.e., for groups of the style used prior to HDF5 Release 1.8.0.

See `H5Pset_local_heap_size_hint` for further discussion.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
size_t *`size_hint`	OUT: Anticipated size of local heap

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE h5pget_local_heap_size_hint_f(gcpl_id, size_hint, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id
                                  ! Group creation property list identifier
  INTEGER(SIZE_T), INTENT(OUT) :: size_hint
                                  ! Hint for size of local heap
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
                                  ! 0 on success and -1 on failure
END SUBROUTINE h5pget_local_heap_size_hint_f
```

History:

Release	**Change**
1.8.0	Function introduced in this release.

Name: H5Pget_mdc_config
Signature:

herr_t `H5Pget_mdc_config`(*hid_t* `plist_id`, *H5AC_cache_config_t* *`config_ptr`)

Purpose:

Get the current initial metadata cache configuration from the indicated File Access Property List.

Description:

`H5Pget_mdc_config` gets the initial metadata cache configuration contained in a file access property list and loads it into the instance of `H5AC_cache_config_t` pointed to by the `config_ptr` parameter. This configuration is used when the file is opened.

Note that the version field of `*config_ptr` must be initialized; this allows the library to support old versions of the `H5AC_cache_config_t` structure.

See the overview of the metadata cache in the special topics section of the user guide for details on the configuration data returned. If you haven't read and understood that documentation, the results of this call will not make much sense.

Parameters:

hid_t `plist_id`	IN: Identifier of the file access property list.
H5AC_cache_config_t *`config_ptr`	IN/OUT: Pointer to the instance of H5AC_cache_config_t in which the current metadata cache configuration is to be reported. The fields of this structure are discussed below:
General configuration section:	
int `version`	IN: Integer field indicating the the version of the H5AC_cache_config_t in use. This field should be set to H5AC__CURR_CACHE_CONFIG_VERSION (defined in H5ACpublic.h).
hbool_t `rpt_fcn_enabled`	OUT: Boolean flag indicating whether the adaptive cache resize report function is enabled. This field should almost always be set to FALSE. Since resize algorithm activity is reported via stdout, it MUST be set to FALSE on Windows machines. The report function is not supported code, and can be expected to change between versions of the library. Use it at your own risk.
hbool_t `open_trace_file`	OUT: Boolean field indicating whether the `trace_file_name` field should be used to open a trace file for the cache. This field will always be set to `FALSE` in this context.
hbool_t `close_trace_file`	OUT: Boolean field indicating whether the current trace file (if any) should be closed. This field will always be set to `FALSE` in this

	context.
char *`trace_file_name`	OUT: Full path name of the trace file to be opened if the `open_trace_file` field is `TRUE`. This field will always be set to the empty string in this context.
hbool_t `evictions_enabled`	OUT: Boolean flag indicating whether metadata cache entry evictions will be enabled when the file is opened / created.
hbool_t `set_initial_size`	OUT: Boolean flag indicating whether the cache should be created with a user specified initial maximum size.
size_t `initial_size`	OUT: Initial maximum size of the cache in bytes, if applicable.
double `min_clean_fraction`	OUT: Float value specifing the minimum fraction of the cache that must be kept either clean or empty when possible.
size_t `max_size`	OUT: Upper bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
size_t `min_size`	OUT: Lower bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
int `epoch_length`	OUT: Number of cache accesses between runs of the adaptive cache resize code.
Increment configuration section:	
enum H5C_cache_incr_mode `incr_mode`	OUT: Enumerated value indicating the operational mode of the automatic cache size increase code. At present, only the following values are legal: H5C_incr__off: Automatic cache size increase is disabled. H5C_incr__threshold: Automatic cache size increase is enabled using the hit rate threshold algorithm.
double `lower_hr_threshold`	OUT: Hit rate threshold used in the hit rate threshold cache size increase algorithm.
double `increment`	OUT: The factor by which the current maximum cache size is multiplied to obtain an initial new maximum cache size if a size increase is triggered in the hit rate threshold cache size increase algorithm.
hbool_t `apply_max_increment`	OUT: Boolean flag indicating whether an upper limit will be applied to the size of cache size increases.

size_t `max_increment`	OUT: The maximum number of bytes by which the maximum cache size can be increased in a single step -- if applicable.
enum H5C_cache_flash_incr_mode `flash_incr_mode`	OUT: Enumerated value indicating the operational mode of the flash cache size increase code. At present, only the following values are legal: H5C_flash_incr__off: Flash cache size increase is disabled. H5C_flash_incr__add_space: Flash cache size increase is enabled using the add space algorithm.
double `flash_threshold`	OUT: The factor by which the current maximum cache size is multiplied to obtain the minimum size entry / entry size increase which may trigger a flash cache size increase.
double `flash_multiple`	OUT: The factor by which the size of the triggering entry / entry size increase is multiplied to obtain the initial cache size increment. This increment may be reduced to reflect existing free space in the cache and the `max_size` field above.
Decrement configuration section:	
enum H5C_cache_decr_mode `decr_mode`	OUT: Enumerated value indicating the operational mode of the automatic cache size decrease code. At present, the following values are legal: H5C_decr__off: Automatic cache size decrease is disabled, and the remaining decrement fields are ignored. H5C_decr__threshold: Automatic cache size decrease is enabled using the hit rate threshold algorithm. H5C_decr__age_out: Automatic cache size decrease is enabled using the ageout algorithm. H5C_decr__age_out_with_threshold: Automatic cache size decrease is enabled using the ageout with hit rate threshold algorithm
double `upper_hr_threshold`	OUT: Upper hit rate threshold. This value is only used if the decr_mode is either H5C_decr__threshold or H5C_decr__age_out_with_threshold.

double `decrement`	OUT: Factor by which the current max cache size is multiplied to obtain an initial value for the new cache size when cache size reduction is triggered in the hit rate threshold cache size reduction algorithm.
hbool_t `apply_max_decrement`	OUT: Boolean flag indicating whether an upper limit should be applied to the size of cache size decreases.
size_t `max_decrement`	OUT: The maximum number of bytes by which cache size can be decreased if any single step, if applicable.
int `epochs_before_eviction`	OUT: The minimum number of epochs that an entry must reside unaccessed in cache before being evicted under either of the ageout cache size reduction algorithms.
hbool_t `apply_empty_reserve`	OUT: Boolean flag indicating whether an empty reserve should be maintained under either of the ageout cache size reduction algorithms.
double `empty_reserve`	OUT: Empty reserve for use with the ageout cache size reduction algorithms, if applicable.
Parallel configuration section:	
int `dirty_bytes_threshold`	OUT: Threshold number of bytes of dirty metadata generation for triggering synchronizations of the metadata caches serving the target file in the parallel case. Synchronization occurs whenever the number of bytes of dirty metadata created since the last synchronization exceeds this limit.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Name: H5Pget_meta_block_size
Signature:

herr_t `H5Pget_meta_block_size`(*hid_t* `fapl_id`, *hsize_t* *`size`)

Purpose:

Returns the current metadata block size setting.

Description:

`H5Pget_meta_block_size` returns the current minimum size, in bytes, of new metadata block allocations. This setting is retrieved from the file access property list `fapl_id`.

This value is set by H5Pset_meta_block_size and is retrieved from the file access property list `fapl_id`.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t *`size`	OUT: Minimum size, in bytes, of metadata block allocations.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_meta_block_size_f

```
SUBROUTINE h5pget_meta_block_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! File access property list
                                         ! identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: size  ! Metadata block size
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pget_meta_block_size_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pget_multi_type

Signature:

herr_t `H5Pget_multi_type` (*hid_t* `fapl_id`, *H5FD_mem_t* `*type`)

Purpose:

Retrieves type of data property for `MULTI` driver.

Description:

`H5Pget_multi_type` retrieves the type of data setting from the file access or data transfer property list `fapl_id`. This enables a user application to specify the type of data the application wishes to access so that the application can retrieve a file handle for low-level access to the particular member of a set of `MULTI` files in which that type of data is stored. The file handle is retrieved with a separate call to `H5Fget_vfd_handle` (or, in special circumstances, to `H5FDget_vfd_handle`; see *Virtual File Layer* and *List of VFL Functions* in *HDF5 Technical Notes*).

The type of data returned in `type` will be one of those listed in the discussion of the `type` parameter in the the description of the function `H5Pset_multi_type`.

Use of this function is only appropriate for an HDF5 file written as a set of files with the `MULTI` file driver.

Parameters:

hid_t `fapl_id`	IN: File access property list or data transfer property list identifier.
H5FD_mem_t `*type`	OUT: Type of data.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pget_nfilters

Signature:

int `H5Pget_nfilters`(*hid_t* `plist`)

Purpose:

Returns the number of filters in the pipeline.

Description:

`H5Pget_nfilters` returns the number of filters defined in the filter pipeline associated with the property list `plist`.

In each pipeline, the filters are numbered from 0 through *N*-1, where *N* is the value returned by this function. During output to the file, the filters are applied in increasing order; during input from the file, they are applied in decreasing order.

`H5Pget_nfilters` returns the number of filters in the pipeline, including zero (`0`) if there are none.

Note:

This function currently supports only the permanent filter pipeline; `plist_id` must be a dataset creation property list.

Parameters:

hid_t `plist`	IN: Property list identifier.

Returns:

Returns the number of filters in the pipeline if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_nfilters_f

```
SUBROUTINE h5pget_nfilters_f(prp_id, nfilters, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id   ! Dataset creation property
                                         ! list identifier
  INTEGER, INTENT(OUT) :: nfilters       ! The number of filters in
                                         ! the pipeline
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pget_nfilters_f
```

Name: H5Pget_nlinks
Signature:

herr_t `H5Pget_nlinks`(*hid_t* `lapl_id`, *size_t* *`nlinks`)

Purpose:

Retrieves the maximum number of link traversals.

Description:

`H5Pget_nlinks` retrieves the maximum number of soft or user-defined link traversals allowed, `nlinks`, before the library assumes it has found a cycle and aborts the traversal. This value is retrieved from the link access property list `lapl_id`.

The limit on the number soft or user-defined link traversals is designed to terminate link traversal if one or more links form a cycle. User control is provided because some files may have legitimate paths formed of large numbers of soft or user-defined links. This property can be used to allow traversal of as many links as desired.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
size_t *`nlinks`	OUT: Maximum number of links to traverse

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_nlinks_f

```
SUBROUTINE h5pget_nlinks_f(lapl_id, nlinks, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: lapl_id
                                    ! File access property list identifier
  INTEGER(SIZE_T), INTENT(OUT) :: nlinks
                                    ! Maximum number of links to traverse
  INTEGER, INTENT(OUT) :: hdferr     ! Error code
                                    ! 0 on success and -1 on failure
END SUBROUTINE h5pget_nlinks_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_nprops

Signature:

int `H5Pget_nprops`(*hid_t* `id`, *size_t* `*nprops`)

Purpose:

Queries number of properties in property list or class.

Description:

`H5Pget_nprops` retrieves the number of properties in a property list or class. If a property class identifier is given, the number of registered properties in the class is returned in `nprops`. If a property list identifier is given, the current number of properties in the list is returned in `nprops`.

Parameters:

hid_t `id`	IN: Identifier of property object to query
size_t `*nprops`	OUT: Number of properties in object

Returns:

Success: a non-negative value

Failure: a negative value

Fortran90 Interface: h5pget_nprops_f

```
SUBROUTINE h5pget_nprops_f(prp_id, nprops, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id    ! Property list identifier
  INTEGER(SIZE_T), INTENT(OUT) :: nprops  ! Number of properties
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_nprops_f
```

Name: H5Pget_preserve

Signature:

int `H5Pget_preserve`(*hid_t* `plist`)

Purpose:

Checks status of the dataset transfer property list.

Notice:

This function is deprecated as it is no longer useful; compound datatype field preservation is now core functionality in the HDF5 Library.

Description:

`H5Pget_preserve` checks the status of the dataset transfer property list.

Parameters:

hid_t `plist`	IN: Identifier for the dataset transfer property list.

Returns:

Returns TRUE or FALSE if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_preserve_f

```
SUBROUTINE h5pget_preserve_f(prp_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id   ! Dataset transfer property
                                         ! list identifier
  LOGICAL, INTENT(OUT)       :: flag     ! Status of for the dataset
                                         ! transfer property list
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pget_preserve_f
```

History:

Release	**Fortran90**
1.6.0	The `flag` parameter was changed from *INTEGER* to *LOGICAL* to better match the C API.

Name: H5Pget_obj_track_times

Signature:

herr_t `H5Pget_obj_track_times`(*hid_t* `ocpl_id`, *hbool_t* `*track_times`)

Purpose:

Determines whether times associated with an object are being recorded.

Description:

`H5get_obj_track_times` queries the object creation property list, `ocpl_id`, to determine whether object times are being recorded.

If `track_times` is returned as `TRUE`, times are being recorded; if `track_times` is returned as `FALSE`, times are not being recorded.

Time data can be retrieved with `H5Oget_info`, which will return it in the `H5O_info_t` struct.

If times are not tracked, they will be reported as follows when queried:

```
12:00 AM UDT, Jan. 1, 1970
```

See `H5Pset_obj_track_times` for further discussion.

Parameters:

hid_t `ocpl_id`	IN: Object creation property list identifier
hbool_t `*track_times`	OUT: Boolean value, `TRUE` or `FALSE`, specifying whether object times are being recorded

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_obj_track_times_f

```
SUBROUTINE h5pget_obj_track_times_f(plist_id, flag, hdferr)
    IMPLICIT NONE
    INTEGER(HID_T), INTENT(IN) :: plist_id
                                   ! Dataset creation property
                                   ! list identifier
    LOGICAL, INTENT(OUT) :: flag   ! Object timestamp setting
    INTEGER, INTENT(OUT) :: hdferr ! Error code
                                   ! 0 on success and -1 on failure
END SUBROUTINE h5pget_obj_track_times_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_shared_mesg_index

Signature:

herr_t `H5Pget_shared_mesg_index`(*hid_t* `fcpl_id`, *unsigned* `index_num`, *unsigned* *`mesg_type_flags`, *unsigned* *`min_mesg_size`)

Purpose:

Retrieves the configuration settings for a shared message index.

Description:

`H5Pget_shared_mesg_index` retrieves the message type and minimum message size settings from the file creation property list `fcpl_id` for the shared object header message index specified by `index_num`.

`index_num` specifies the index. `index_num` is zero-indexed, so in a file with three indexes, they will be numbered `0`, `1`, and `2`.

`mesg_type_flags` and `min_mesg_size` will contain, respectively, the types of messages and the minimum size, in bytes, of messages that can be stored in this index.

Valid message types are described in `H5Pset_shared_mesg_index`.

Parameters:

hid_t `fcpl_id`	IN: File creation property list identifier.
unsigned `index_num`	IN: Index being configured.
unsigned *`mesg_type_flags`	OUT: Types of messages that may be stored in this index.
unsigned *`min_mesg_size`	OUT: Minimum message size.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_shared_mesg_nindexes

Signature:

herr_t `H5Pget_shared_mesg_nindexes`(*hid_t* `fcpl_id`, *unsigned* `*nindexes`)

Purpose:

Retrieves number of shared object header message indexes in file creation property list.

Description:

`H5Pget_shared_mesg_nindexes` retrieves the number of shared object header message indexes in the specified file creation property list `fcpl_id`.

If the value of `nindexes` is `0` (zero), shared object header messages are disabled in files created with this property list.

Parameters:

hid_t `fcpl_id`	IN: File creation property list
unsigned `*nindexes`	OUT: Number of shared object header message indexes available in files created with this property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_shared_mesg_phase_change

Signature:

herr_t `H5Pget_shared_mesg_phase_change(` *hid_t* `fcpl_id`, *unsigned* `*max_list`, *unsigned* `*min_btree`)

Purpose:

Retrieves shared object header message phase change information.

Description:

`H5Pget_shared_mesg_phase_change` retrieves the threshold values for storage of shared object header message indexes in a file. These phase change thresholds determine the point at which the index storage mechanism changes from a more compact list format to a more performance-oriented B-tree format, and vice-versa.

By default, a shared object header message index is initially stored as a compact list. When the number of messages in an index exceeds the specified `max_list` threshold, storage switches to a B-tree format for impoved performance. If the number of messages subsequently falls below the `min_btree` threshold, the index will revert to the list format.

If `max_compact` is set to `0` (zero), shared object header message indexes in the file will always be stored as B-trees.

`fcpl_id` specifies the file creation property list.

Parameters:

hid_t `fcpl_id`	IN: File creation property list identifier
unsigned `*max_compact`	OUT: Threshold above which storage of a shared object header message index shifts from list to B-tree
unsigned `*min_btree`	OUT: Threshold below which storage of a shared object header message index reverts to list format

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pget_sieve_buf_size

Signature:

herr_t `H5Pget_sieve_buf_size`(*hid_t* `fapl_id`, *hsize_t* `*size`)

Purpose:

Returns maximum data sieve buffer size.

Description:

`H5Pget_sieve_buf_size` retrieves, `size`, the current maximum size of the data sieve buffer.

This value is set by H5Pset_sieve_buf_size and is retrieved from the file access property list `fapl_id`.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t `*size`	IN: Maximum size, in bytes, of data sieve buffer.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pget_sieve_buf_size_f

```
SUBROUTINE h5pget_sieve_buf_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! File access property list
                                          ! identifier
  INTEGER(SIZE_T), INTENT(OUT) :: size   ! Sieve buffer size
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_sieve_buf_size_f
```

History:

Release	**C**
1.6.0	The `size` parameter has changed from type *hsize_t* to *size_t*.
1.4.0	Function introduced in this release.

Name: H5Pget_size

Signature:

int `H5Pget_size`(*hid_t* `id`, *const char* *`name`, *size_t* *`size`)

Purpose:

Queries the size of a property value in bytes.

Description:

`H5Pget_size` retrieves the size of a property's value in bytes. This function operates on both property lists and property classes

Zero-sized properties are allowed and return `0`.

Parameters:

hid_t `id`	IN: Identifier of property object to query
const char *`name`	IN: Name of property to query
size_t *`size`	OUT: Size of property in bytes

Returns:

Success: a non-negative value

Failure: a negative value

Fortran90 Interface: h5pget_size_f

```
SUBROUTINE h5pget_size_f(prp_id, name, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to query
  INTEGER(SIZE_T), INTENT(OUT) :: size  ! Size in bytes
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pget_size_f
```

Name: H5Pget_sizes

Signature:

herr_t `H5Pget_sizes`(*hid_t* `plist`, *size_t* * `sizeof_addr`, *size_t* * `sizeof_size`)

Purpose:

Retrieves the size of the offsets and lengths used in an HDF5 file.

Description:

`H5Pget_sizes` retrieves the size of the offsets and lengths used in an HDF5 file. This function is only valid for file creation property lists.

Parameters:

hid_t `plist`	IN: Identifier of property list to query.
size_t * `size`	OUT: Pointer to location to return offset size in bytes.
size_t * `size`	OUT: Pointer to location to return length size in bytes.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_sizes_f

```
SUBROUTINE h5pget_sizes_f(prp_id, sizeof_addr, sizeof_size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER(SIZE_T), DIMENSION(:), INTENT(OUT) :: sizeof_addr
                                        ! Size of an object address in bytes
  INTEGER(SIZE_T), DIMENSION(:), INTENT(OUT) :: sizeof_size
                                        ! Size of an object in bytes
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pget_sizes_f
```

Name: H5Pget_small_data_block_size

Signature:

herr_t `H5Pget_small_data_block_size`(*hid_t* `fapl_id`, *hsize_t* *`size`)

Purpose:

Retrieves the current small data block size setting.

Description:

`H5Pget_small_data_block_size` retrieves the current setting for the size of the small data block.

If the returned value is zero (`0`), the small data block mechanism has been disabled for the file.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t *`size`	OUT: Maximum size, in bytes, of the small data block.

Returns:

Returns a non-negative value if successful; otherwise a negative value.

Fortran90 Interface: h5pget_small_data_block_size_f

```
SUBROUTINE h5pget_small_data_block_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id  ! File access property list
                                          ! identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: size   ! Small raw data block size
  INTEGER, INTENT(OUT)       :: hdferr    ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_small_data_block_size_f
```

History:

Release	**C**
1.4.4	Function introduced in this release.

Name: H5Pget_sym_k

Signature:

herr_t `H5Pget_sym_k`(*hid_t* `plist`, *unsigned* * `ik`, *unsigned* * `lk`)

Purpose:

Retrieves the size of the symbol table B-tree 1/2 rank and the symbol table leaf node 1/2 size.

Description:

`H5Pget_sym_k` retrieves the size of the symbol table B-tree 1/2 rank and the symbol table leaf node 1/2 size. This function is only valid for file creation property lists. If a parameter valued is set to NULL, that parameter is not retrieved. See the description for H5Pset_sym_k for more information.

Parameters:

hid_t `plist`	IN: Property list to query.
unsigned * `ik`	OUT: Pointer to location to return the symbol table's B-tree 1/2 rank.
unsigned * `size`	OUT: Pointer to location to return the symbol table's leaf node 1/2 size.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_sym_k_f

```
SUBROUTINE h5pget_sym_k_f(prp_id, ik, lk, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(OUT) :: ik            ! Symbol table tree rank
  INTEGER, INTENT(OUT) :: lk            ! Symbol table node size
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pget_sym_k_f
```

History:

Release	**C**
1.6.4	`ik` parameter type changed to *unsigned*
1.6.0	The `ik` parameter has changed from type *int* to *unsigned*

Name: H5Pget_type_conv_cb

Signature:

herr_t `H5Pget_type_conv_cb`(*hid_t* `plist`, *H5T_conv_except_func_t* *`func`, *void* **`op_data`)

Purpose:

Gets user-defined datatype conversion callback function.

Description:

`H5Pget_type_conv_cb` gets the user-defined datatype conversion callback function `func` in the dataset transfer property list `plist`.

The parameter `op_data` is a pointer to user-defined input data for the callback function.

The callback function `func` defines the actions an application is to take when there is an exception during datatype conversion.

Please refer to the function `H5Pset_type_conv_cb` for more details.

Parameters:

hid_t `plist`	IN: Dataset transfer property list identifier.
H5T_conv_except_func_t *`func`	OUT: User-defined type conversion callback function.
void **`op_data`	OUT: User-defined input data for the callback function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

Name: H5Pget_userblock

Signature:

herr_t `H5Pget_userblock`(*hid_t* `plist`, *hsize_t* * `size`)

Purpose:

Retrieves the size of a user block.

Description:

`H5Pget_userblock` retrieves the size of a user block in a file creation property list.

Parameters:

hid_t `plist`	IN: Identifier for property list to query.
hsize_t * `size`	OUT: Pointer to location to return user-block size.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_userblock_f

```
SUBROUTINE h5pget_userblock_f(prp_id, block_size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id   ! Property list identifier
  INTEGER(HSIZE_T), DIMENSION(:), INTENT(OUT) ::  block_size
                                          ! Size of the user-block in bytes
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pget_userblock_f
```

Name: H5Pget_version

Signature:

herr_t `H5Pget_version`(*hid_t* `plist`, *unsigned* * `super`, *unsigned* * `freelist`, *unsigned* * `stab`, *unsigned* * `shhdr`)

Purpose:

Retrieves the version information of various objects for a file creation property list.

Description:

`H5Pget_version` retrieves the version information of various objects for a file creation property list. Any pointer parameters which are passed as NULL are not queried.

Parameters:

hid_t `plist`	IN: Identifier of the file creation property list.
unsigned * `super`	OUT: Pointer to location to return super block version number.
unsigned * `freelist`	OUT: Pointer to location to return global freelist version number.
unsigned * `stab`	OUT: Pointer to location to return symbol table version number.
unsigned * `shhdr`	OUT: Pointer to location to return shared object header version number.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_version_f

```
SUBROUTINE h5pget_version_f(prp_id, boot, freelist, &
                            stab, shhdr, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id         ! Property list identifier
  INTEGER, DIMENSION(:), INTENT(OUT) :: boot   ! Array to put boot block
                                               ! version number
  INTEGER, DIMENSION(:), INTENT(OUT) :: freelist
                                               ! Array to put global
                                               ! freelist version number
  INTEGER, DIMENSION(:), INTENT(OUT) :: stab   ! Array to put symbol table
                                               ! version number
  INTEGER, DIMENSION(:), INTENT(OUT) :: shhdr  ! Array to put shared object
                                               ! header version number
  INTEGER, INTENT(OUT) :: hdferr               ! Error code
                                               ! 0 on success and -1 on failure
END SUBROUTINE h5pget_version_f
```

History:

Release	**C**
1.6.4	`boot`, `freelist`, `stab`, `shhdr` parameter types changed to *unsigned*.

Name: H5Pget_vlen_mem_manager

Signature:

herr_t `H5Pget_vlen_mem_manager`(*hid_t* `plist`, *H5MM_allocate_t* *`alloc`, *void* **`alloc_info`, *H5MM_free_t* *`free`, *void* **`free_info`)

Purpose:

Gets the memory manager for variable-length datatype allocation in `H5Dread` and `H5Dvlen_reclaim`.

Description:

`H5Pget_vlen_mem_manager` is the companion function to `H5Pset_vlen_mem_manager`, returning the parameters set by that function.

Parameters:

hid_t `plist`	IN: Identifier for the dataset transfer property list.
H5MM_allocate_t `alloc`	OUT: User's allocate routine, or `NULL` for system `malloc`.
void *`alloc_info`	OUT: Extra parameter for user's allocation routine. Contents are ignored if preceding parameter is `NULL`.
H5MM_free_t `free`	OUT: User's free routine, or `NULL` for system `free`.
void *`free_info`	OUT: Extra parameter for user's free routine. Contents are ignored if preceding parameter is `NULL`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

Name: H5Pinsert

Signatures:

herr_t `H5Pinsert`(*hid_t* `plid`, *const char* *`name`, *size_t* `size`, *void* *`value`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_close_func_t* `close`) [1]

herr_t `H5Pinsert`(*hid_t* `plid`, *const char* *`name`, *size_t* `size`, *void* *`value`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_compare_func_t* `compare`, *H5P_prp_close_func_t* `close`) [2]

Purpose:

Registers a temporary property with a property list.

Description:

`H5Pinsert` is a macro that is mapped to either `H5Pinsert1` or `H5Pinsert2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

- ◊ The `H5Pinsert` macro will be mapped to `H5Pinsert1` and will use the `H5Pinsert1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.
- ◊ The `H5Pinsert` macro mapped to `H5Pinsert2` and will use the `H5Pinsert2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in “API Compatibility Macros in HDF5” we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Pinsert` is mapped to the most recent version of the function, currently `H5Pinsert2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Pinsert` will be mapped to `H5Pinsert1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Pinsert` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Pinsert2`
Enable deprecated symbols	`H5Pinsert2`
Disable deprecated symbols	`H5Pinsert2`
Emulate Release 1.6 interface	`H5Pinsert1`

Function-level macros

`H5Pinsert_vers = 2`	`H5Pinsert2`
`H5Pinsert_vers = 1`	`H5Pinsert1`

Interface history: Signature [1] above is the original `H5Pinsert` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Pinsert1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Pinsert2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface: h5pinsert_f

```
SUBROUTINE h5pinsert_f
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist   ! Property list class identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to insert
  INTEGER(SIZE_T), INTENT(IN) :: size   ! Size of the property value
  TYPE,   INTENT(IN) :: value           ! Property value
                                        ! Supported types are:
                                        !    INTEGER
                                        !    REAL
                                        !    DOUBLE PRECISION
                                        !    CHARACTER(LEN=*)
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pinsert_f
```

History:

Release	**C**
1.8.0	The function `H5Pinsert` renamed to `H5Pinsert1` and deprecated in this release. The macro `H5Pinsert` and the function `H5Pinsert2` introduced in this release.

Name: H5Pinsert1

Signature:

herr_t `H5Pinsert1`(*hid_t* `plid`, *const char* *`name`, *size_t* `size`, *void* *`value`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_close_func_t* `close`)

Purpose:

Registers a temporary property with a property list.

Notice:

This function is renamed from `H5Pinsert` *and deprecated in favor of the function* `H5Pinsert2` *or the new macro* `H5Pinsert`.

Description:

`H5Pinsert1` create a new property in a property list. The property will exist only in this property list and copies made from it.

The initial property value must be provided in `value` and the property value will be set accordingly.

The name of the property must not already exist in this list, or this routine will fail.

The `set` and `get` callback routines may be set to NULL if they are not needed.

Zero-sized properties are allowed and do not store any data in the property list. The default value of a zero-size property may be set to NULL. They may be used to indicate the presence or absence of a particular piece of information.

The `set` routine is called before a new value is copied into the property. The `H5P_prp_set_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_set_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`new_value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being modified
const char *`name`	IN: The name of the property being modified
size_t `size`	IN: The size of the property in bytes
void **`new_value`	IN: Pointer to new value pointer for the property being modified

The `set` routine may modify the value pointer to be set and those changes will be used when setting the property's value. If the `set` routine returns a negative value, the new property value is not copied into the property and the set routine returns an error value. The `set` routine will be called for the initial value.

Note: The `set` callback function may be useful to range check the value being set for the property or may perform some transformation or translation of the value set. The `get` callback would then reverse the transformation or translation. A single `get` or `set` callback could handle multiple properties by performing different actions based on the property name or other properties in the property list.

The `get` routine is called when a value is retrieved from a property value. The `H5P_prp_get_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_get_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`);

The parameters to the above callback function are:

hid_t `prop_id`	IN: The identifier of the property list being queried
const char *`name`	IN: The name of the property being queried
size_t `size`	IN: The size of the property in bytes
void *`value`	IN: The value of the property being returned

The `get` routine may modify the value to be returned from the query and those changes will be preserved. If the `get` routine returns a negative value, the query routine returns an error value.

The `delete` routine is called when a property is being deleted from a property list. The `H5P_prp_delete_func_t` callback function is defined as follows:

`typedef herr_t` (*`H5P_prp_delete_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`);

The parameters to the above callback function are:

hid_t `prop_id`	IN: The identifier of the property list the property is being deleted from
const char * `name`	IN: The name of the property in the list
size_t `size`	IN: The size of the property in bytes
void * `value`	IN: The value for the property being deleted

The `delete` routine may modify the value passed in, but the value is not used by the library when the `delete` routine returns. If the `delete` routine returns a negative value, the property list delete routine returns an error value but the property is still deleted.

The `copy` routine is called when a new property list with this property is being created through a copy operation. The `H5P_prp_copy_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_copy_func_t`)(*const char* *`name`, *size_t* `size`, *void* *`value`);

The parameters to the above callback function are:

const char *`name`	IN: The name of the property being copied
size_t `size`	IN: The size of the property in bytes
void * `value`	IN/OUT: The value for the property being copied

The `copy` routine may modify the value to be set and those changes will be stored as the new value of the property. If the `copy` routine returns a negative value, the new property value is not copied into the property and the copy routine returns an error value.

The `close` routine is called when a property list with this property is being closed. The `H5P_prp_close_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_close_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being closed
`const char *`*name*	IN: The name of the property in the list

`size_t` *size*	IN: The size of the property in bytes
`void *`*value*	IN: The value for the property being closed

The `close` routine may modify the value passed in, the value is not used by the library when the `close` routine returns. If the `close` routine returns a negative value, the property list close routine returns an error value but the property list is still closed.

Note: There is no `create` callback routine for temporary property list objects; the initial value is assumed to have any necessary setup already performed on it.

Parameters:

hid_t `plid`	IN: Property list identifier to create temporary property within
const char *`name`	IN: Name of property to create
size_t `size`	IN: Size of property in bytes
void *`value`	IN: Initial value for the property
H5P_prp_set_func_t `set`	IN: Callback routine called before a new value is copied into the property's value
H5P_prp_get_func_t `get`	IN: Callback routine called when a property value is retrieved from the property
H5P_prp_delete_func_t `delete`	IN: Callback routine called when a property is deleted from a property list
H5P_prp_copy_func_t `copy`	IN: Callback routine called when a property is copied from an existing property list
H5P_prp_close_func_t `close`	IN: Callback routine called when a property list is being closed and the property value will be disposed of

Returns:

Success: a non-negative value
Failure: a negative value

Fortran90 Interface: h5pinsert_f

```
SUBROUTINE h5pinsert_f
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist   ! Property list class identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to insert
  INTEGER(SIZE_T), INTENT(IN) :: size   ! Size of the property value
  TYPE,   INTENT(IN) :: value           ! Property value
                                        ! Supported types are:
                                        !    INTEGER
                                        !    REAL
                                        !    DOUBLE PRECISION
                                        !    CHARACTER(LEN=*)
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pinsert_f
```

History:

Release	**C**
1.8.0	Function `H5Pinsert` renamed to `H5Pinsert1` and deprecated in this release.

Name: H5Pinsert2

Signature:

herr_t `H5Pinsert2`(*hid_t* `plid`, *const char* *`name`, *size_t* `size`, *void* *`value`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_compare_func_t* `compare`, *H5P_prp_close_func_t* `close`)

Purpose:

Registers a temporary property with a property list.

Description:

`H5Pinsert2` create a new property in a property list. The property will exist only in this property list and copies made from it.

The initial property value must be provided in `value` and the property value will be set accordingly.

The name of the property must not already exist in this list, or this routine will fail.

The `set` and `get` callback routines may be set to NULL if they are not needed.

Zero-sized properties are allowed and do not store any data in the property list. The default value of a zero-size property may be set to NULL. They may be used to indicate the presence or absence of a particular piece of information.

The `set` routine is called before a new value is copied into the property. The `H5P_prp_set_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_set_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`new_value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being modified
const char *`name`	IN: The name of the property being modified
size_t `size`	IN: The size of the property in bytes
void **`new_value`	IN: Pointer to new value pointer for the property being modified

The `set` routine may modify the value pointer to be set and those changes will be used when setting the property's value. If the `set` routine returns a negative value, the new property value is not copied into the property and the set routine returns an error value. The `set` routine will be called for the initial value.

Note: The `set` callback function may be useful to range check the value being set for the property or may perform some transformation or translation of the value set. The `get` callback would then reverse the transformation or translation. A single `get` or `set` callback could handle multiple properties by performing different actions based on the property name or other properties in the property list.

The `get` routine is called when a value is retrieved from a property value. The `H5P_prp_get_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_get_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`);

The parameters to the above callback function are:

hid_t `prop_id`	IN: The identifier of the property list being queried
const char *`name`	IN: The name of the property being queried
size_t `size`	IN: The size of the property in bytes
void *`value`	IN: The value of the property being returned

The `get` routine may modify the value to be returned from the query and those changes will be preserved. If the `get` routine returns a negative value, the query routine returns an error value.

The `delete` routine is called when a property is being deleted from a property list. The `H5P_prp_delete_func_t` callback function is defined as follows:

`typedef herr_t` (*`H5P_prp_delete_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`);

The parameters to the above callback function are:

hid_t `prop_id`	IN: The identifier of the property list the property is being deleted from
const char * `name`	IN: The name of the property in the list
size_t `size`	IN: The size of the property in bytes
void * `value`	IN: The value for the property being deleted

The `delete` routine may modify the value passed in, but the value is not used by the library when the `delete` routine returns. If the `delete` routine returns a negative value, the property list delete routine returns an error value but the property is still deleted.

The `copy` routine is called when a new property list with this property is being created through a copy operation. The `H5P_prp_copy_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_copy_func_t`)(*const char* *`name`, *size_t* `size`, *void* *`value`);

The parameters to the above callback function are:

const char *`name`	IN: The name of the property being copied
size_t `size`	IN: The size of the property in bytes
void * `value`	IN/OUT: The value for the property being copied

The `copy` routine may modify the value to be set and those changes will be stored as the new value of the property. If the `copy` routine returns a negative value, the new property value is not copied into the property and the copy routine returns an error value.

The `compare` routine is called when a property list with this property is compared to another property list with the same property. The `H5P_prp_compare_func_t` callback function is defined as follows:

typedef int (*`H5P_prp_compare_func_t`)(*const void* *`value1`, *const void* *`value2`, *size_t* `size`); The parameters to the callback function are defined as follows:

const void *`value1`	IN: The value of the first property to compare
const void *`value2`	IN: The value of the second property to compare
size_t `size`	IN: The size of the property in bytes

The `compare` routine may *not* modify the values. The `compare` routine should return a positive value if `value1` is greater than `value2`, a negative value if `value2` is greater than `value1` and zero if `value1` and `value2` are equal.

The `close` routine is called when a property list with this property is being closed. The `H5P_prp_close_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_close_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being closed
`const char *`*name*	IN: The name of the property in the list
`size_t` *size*	IN: The size of the property in bytes
`void *`*value*	IN: The value for the property being closed

The `close` routine may modify the value passed in, the value is not used by the library when the `close` routine returns. If the `close` routine returns a negative value, the property list close routine returns an error value but the property list is still closed.

Note: There is no `create` callback routine for temporary property list objects; the initial value is assumed to have any necessary setup already performed on it.

Parameters:

hid_t `plid`	IN: Property list identifier to create temporary property within
const char *`name`	IN: Name of property to create
size_t `size`	IN: Size of property in bytes
void *`value`	IN: Initial value for the property
H5P_prp_set_func_t `set`	IN: Callback routine called before a new value is copied into the property's value
H5P_prp_get_func_t `get`	IN: Callback routine called when a property value is retrieved from the property
H5P_prp_delete_func_t `delete`	IN: Callback routine called when a property is deleted from a property list
H5P_prp_copy_func_t `copy`	IN: Callback routine called when a property is copied from an existing property list
H5P_prp_compare_func_t `compare`	IN: Callback routine called when a property is compared with another property list
H5P_prp_close_func_t `close`	IN: Callback routine called when a property list is being closed and the property value will be disposed of

Returns:

Success: a non-negative value
Failure: a negative value

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Last modified: 20 April 2009

Name: H5Pisa_class

Signature:

htri_t `H5Pisa_class`(*hid_t* `plist`, *hid_t* `pclass`)

Purpose:

Determines whether a property list is a member of a class.

Description:

`H5Pisa_class` checks to determine whether the property list `plist` is a member of the property list class `pclass`.

Parameters:

hid_t `plist`	IN: Property list identifier
`hid_t` *pclass*	IN: Property list class identifier

Returns:

Returns a positive value if true or zero if false; returns a negative value on failure.

See Also:

`H5Pcreate`

Fortran90 Interface: h5pisa_class_f

```
SUBROUTINE h5pisa_class_f(plist, pclass, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist     ! Property list identifier
  INTEGER(HID_T), INTENT(IN) :: pclass    ! Class identifier
  LOGICAL, INTENT(OUT) :: flag            ! Logical flag
                                          !    .TRUE. if a member
                                          !    .FALSE. otherwise
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5pisa_class_f
```

Name: H5Piterate

Purpose:

Iterates over properties in a property class or list.

Signature:

int `H5Piterate`(*hid_t* `id`, *int* * `idx`, *H5P_iterate_t* `iter_func`, *void* * `iter_data`)

Description:

`H5Piterate` iterates over the properties in the property object specified in `id`, which may be either a property list or a property class, performing a specified operation on each property in turn.

For each property in the object, `iter_func` and the additional information specified below are passed to the `H5P_iterate_t` operator function.

The iteration begins with the `idx`-th property in the object; the next element to be processed by the operator is returned in `idx`. If `idx` is NULL, the iterator starts at the first property; since no stopping point is returned in this case, the iterator cannot be restarted if one of the calls to its operator returns non-zero.

The prototype for the `H5P_iterate_t` operator is as follows:

typedef herr_t (*`H5P_iterate_t`)(*hid_t* `id`, *const char* *`name`, *void* *`iter_data`)

The operation receives the property list or class identifier for the object being iterated over, `id`, the name of the current property within the object, `name`, and the pointer to the operator data passed in to `H5Piterate`, `iter_data`.

The valid return values from an operator are as follows:

Zero	Causes the iterator to continue, returning zero when all properties have been processed
Positive	Causes the iterator to immediately return that positive value, indicating short-circuit success. The iterator can be restarted at the index of the next property
Negative	Causes the iterator to immediately return that value, indicating failure. The iterator can be restarted at the index of the next property

`H5Piterate` assumes that the properties in the object identified by `id` remain unchanged through the iteration. If the membership changes during the iteration, the function's behavior is undefined.

Parameters:

hid_t `id`	IN: Identifier of property object to iterate over
int * `idx`	IN/OUT: Index of the property to begin with
H5P_iterate_t `iter_func`	IN: Function pointer to function to be called with each property iterated over
void * `iter_data`	IN/OUT: Pointer to iteration data from user

Returns:

Success: the return value of the last call to `iter_func` if it was non-zero; zero if all properties have been processed

Failure: a negative value

Fortran90 Interface:

None.

Name: H5Pmodify_filter

Signature:

herr_t `H5Pmodify_filter`(*hid_t* `plist`, *H5Z_filter_t* `filter_id`, *unsigned int* `flags`, *size_t* `cd_nelmts`, *const unsigned int* `cd_values[]`)

Purpose:

Modifies a filter in the filter pipeline.

Description:

`H5Pmodify_filter` modifies the specified `filter_id` in the filter pipeline. `plist` must be a dataset creation property list and the modified filter will be in a permanent filter pipeline.

The `filter_id`, `flags` `cd_nelmts[]`, and `cd_values` parameters are used in the same manner and accept the same values as described in the discussion of H5Pset_filter.

Note:

This function currently supports only the permanent filter pipeline; `plist_id` must be a dataset creation property list.

Parameters:

hid_t `plist_id`	IN: Property list identifier.
H5Z_filter_t `filter_id`	IN: Filter to be modified.
unsigned int `flags`	IN: Bit vector specifying certain general properties of the filter.
size_t `cd_nelmts`	IN: Number of elements in `cd_values`.
const unsigned int `cd_values[]`	IN: Auxiliary data for the filter.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pmodify_filter_f

```
SUBROUTINE h5pmodify_filter_f(prp_id, filter, flags, cd_nelmts, &
                              cd_values, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id      ! Property list identifier
  INTEGER, INTENT(IN)        :: filter      ! Filter to be modified
  INTEGER, INTENT(IN)        :: flags       ! Bit vector specifying certain
                                            ! general properties of the filter
  INTEGER(SIZE_T), INTENT(IN) :: cd_nelmts  ! Number of elements in cd_values
  INTEGER, DIMENSION(*), INTENT(IN) :: cd_values
                                            ! Auxiliary data for the filter
  INTEGER, INTENT(OUT)       :: hdferr      ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5pmodify_filter_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pregister

Signatures:

herr_t `H5Pregister`(*hid_t* `class`, *const char* * `name`, *size_t* `size`, *void* * `default`, *H5P_prp_create_func_t* `create`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_close_func_t* `close`) [1]

herr_t `H5Pregister`(*hid_t* `class`, *const char* * `name`, *size_t* `size`, *void* * `default`, *H5P_prp_create_func_t* `create`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_compare_func_t* `compare`, *H5P_prp_close_func_t* `close`) [2]

Purpose:

Returns information about the specified filter.

Description:

`H5Pregister` is a macro that is mapped to either `H5Pregister1` or `H5Pregister2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

- ◊ The `H5Pregister` macro will be mapped to `H5Pregister1` and will use the `H5Pregister1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.
- ◊ The `H5Pregister` macro mapped to `H5Pregister2` and will use the `H5Pregister2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in “API Compatibility Macros in HDF5” we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Pregister` is mapped to the most recent version of the function, currently `H5Pregister2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Pregister` will be mapped to `H5Pregister1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Pregister` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Pregister2`
Enable deprecated symbols	`H5Pregister2`
Disable deprecated symbols	`H5Pregister2`
Emulate Release 1.6 interface	`H5Pregister1`

Function-level macros

`H5Pregister_vers = 2`	`H5Pregister2`
`H5Pregister_vers = 1`	`H5Pregister1`

Interface history: Signature [1] above is the original `H5Pregister` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Pregister1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Pregister2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface: h5pregister_f

```
SUBROUTINE h5pregister_f
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: class   ! Property list class identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to register
  INTEGER(SIZE_T), INTENT(IN) :: size   ! Size of the property value
  TYPE,   INTENT(IN) :: value           ! Property value
                                        ! Supported types are:
                                        !    INTEGER
                                        !    REAL
                                        !    DOUBLE PRECISION
                                        !    CHARACTER(LEN=*)
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pregister_f
```

History:

Release	**C**
1.8.0	The function `H5Pregister` renamed to `H5Pregister1` and deprecated in this release. The macro `H5Pregister` and the function `H5Pregister2` introduced in this release.

Name: H5Pregister1

Signature:

herr_t `H5Pregister1`(*hid_t* `class`, *const char* * `name`, *size_t* `size`, *void* * `default`, *H5P_prp_create_func_t* `create`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_close_func_t* `close`)

Purpose:

Registers a permanent property with a property list class.

Notice:

This function is renamed from `H5Pregister` *and deprecated in favor of the function* `H5Pregister2` *and or the new macro* `H5Pregister`.

Description:

`H5Pregister1` registers a new property with a property list class. The property will exist in all property list objects of `class` created after this routine finishes. The name of the property must not already exist, or this routine will fail. The default property value must be provided and all new property lists created with this property will have the property value set to the default value. Any of the callback routines may be set to NULL if they are not needed.

Zero-sized properties are allowed and do not store any data in the property list. These may be used as flags to indicate the presence or absence of a particular piece of information. The default pointer for a zero-sized property may be set to NULL. The property `create` and `close` callbacks are called for zero-sized properties, but the `set` and `get` callbacks are never called.

The `create` routine is called when a new property list with this property is being created. The `H5P_prp_create_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_create_func_t`)(*const char* *`name`, *size_t* `size`, *void* *`initial_value`); The parameters to this callback function are defined as follows:

const char *`name`	IN: The name of the property being modified
size_t `size`	IN: The size of the property in bytes
void *`initial_value`	IN/OUT: The default value for the property being created, which will be passed to `H5Pregister1`

The `create` routine may modify the value to be set and those changes will be stored as the initial value of the property. If the `create` routine returns a negative value, the new property value is not copied into the property and the create routine returns an error value.

The `set` routine is called before a new value is copied into the property. The `H5P_prp_set_func_t` callback function is defined as follows:

typedef herr_t (*H5P_prp_set_func_t)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`new_value`); The parameters to this callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being modified
const char *`name`	IN: The name of the property being modified
size_t `size`	IN: The size of the property in bytes
void **`new_value`	IN/OUT: Pointer to new value pointer for the property being modified

The `set` routine may modify the value pointer to be set and those changes will be used when setting the property's value. If the `set` routine returns a negative value, the new property value is not copied into the property and the `set` routine returns an error value. The `set` routine will not be called for the initial value, only the `create` routine will be called.

Note: The `set` callback function may be useful to range check the value being set for the property or may perform some transformation or translation of the value set. The `get` callback would then reverse the transformation or translation. A single `get` or `set` callback could handle multiple properties by performing different actions based on the property name or other properties in the property list.

The `get` routine is called when a value is retrieved from a property value. The `H5P_prp_get_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_get_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being queried
const char * `name`	IN: The name of the property being queried
size_t `size`	IN: The size of the property in bytes
void * `value`	IN/OUT: The value of the property being returned

The `get` routine may modify the value to be returned from the query and those changes will be returned to the calling routine. If the `set` routine returns a negative value, the query routine returns an error value.

The `delete` routine is called when a property is being deleted from a property list. The `H5P_prp_delete_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_delete_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list the property is being deleted from
const char * `name`	IN: The name of the property in the list
size_t `size`	IN: The size of the property in bytes
void * `value`	IN: The value for the property being deleted

The `delete` routine may modify the value passed in, but the value is not used by the library when the `delete` routine returns. If the `delete` routine returns a negative value, the property list delete routine returns an error value but the property is still deleted.

The `copy` routine is called when a new property list with this property is being created through a copy operation. The `H5P_prp_copy_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_copy_func_t`)(*const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

const char *`name`	IN: The name of the property being copied
size_t `size`	IN: The size of the property in bytes
void *`value`	IN/OUT: The value for the property being copied

The `copy` routine may modify the value to be set and those changes will be stored as the new value of the property. If the `copy` routine returns a negative value, the new property value is not copied into the property and the copy routine returns an error value.

The `close` routine is called when a property list with this property is being closed. The `H5P_prp_close_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_close_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being closed
const char *`name`	IN: The name of the property in the list
size_t `size`	IN: The size of the property in bytes
void *`value`	IN: The value for the property being closed

The `close` routine may modify the value passed in, but the value is not used by the library when the `close` routine returns. If the `close` routine returns a negative value, the property list close routine returns an error value but the property list is still closed.

Parameters:

hid_t `class`	IN: Property list class to register permanent property within
const char * `name`	IN: Name of property to register
size_t `size`	IN: Size of property in bytes
void * `default`	IN: Default value for property in newly created property lists
H5P_prp_create_func_t `create`	IN: Callback routine called when a property list is being created and the property value will be initialized
H5P_prp_set_func_t `set`	IN: Callback routine called before a new value is copied into the property's value
H5P_prp_get_func_t `get`	IN: Callback routine called when a property value is retrieved from the property
H5P_prp_delete_func_t `delete`	IN: Callback routine called when a property is deleted from a property list
H5P_prp_copy_func_t `copy`	IN: Callback routine called when a property is copied from a property list
H5P_prp_close_func_t `close`	IN: Callback routine called when a property list is being closed and the property value will be disposed of

Returns:

Success: a non-negative value
Failure: a negative value

Fortran90 Interface: h5pregister_f

```
SUBROUTINE h5pregister_f
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: class   ! Property list class identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to register
  INTEGER(SIZE_T), INTENT(IN) :: size   ! Size of the property value
  TYPE,   INTENT(IN) :: value           ! Property value
                                        ! Supported types are:
                                        !    INTEGER
                                        !    REAL
                                        !    DOUBLE PRECISION
                                        !    CHARACTER(LEN=*)
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pregister_f
```

History:

Release	**C**
1.8.0	Function `H5Pregister` renamed to `H5Pregister1` and deprecated in this release.

Name: H5Pregister2

Signature:

herr_t `H5Pregister2`(*hid_t* `class`, *const char* * `name`, *size_t* `size`, *void* * `default`, *H5P_prp_create_func_t* `create`, *H5P_prp_set_func_t* `set`, *H5P_prp_get_func_t* `get`, *H5P_prp_delete_func_t* `delete`, *H5P_prp_copy_func_t* `copy`, *H5P_prp_compare_func_t* `compare`, *H5P_prp_close_func_t* `close`)

Purpose:

Registers a permanent property with a property list class.

Description:

`H5Pregister2` registers a new property with a property list class. The property will exist in all property list objects of `class` created after this routine finishes. The name of the property must not already exist, or this routine will fail. The default property value must be provided and all new property lists created with this property will have the property value set to the default value. Any of the callback routines may be set to NULL if they are not needed.

Zero-sized properties are allowed and do not store any data in the property list. These may be used as flags to indicate the presence or absence of a particular piece of information. The default pointer for a zero-sized property may be set to NULL. The property `create` and `close` callbacks are called for zero-sized properties, but the `set` and `get` callbacks are never called.

The `create` routine is called when a new property list with this property is being created. The `H5P_prp_create_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_create_func_t`)(*const char* *`name`, *size_t* `size`, *void* *`initial_value`); The parameters to this callback function are defined as follows:

const char *`name`	IN: The name of the property being modified
size_t `size`	IN: The size of the property in bytes
void *`initial_value`	IN/OUT: The default value for the property being created, which will be passed to `H5Pregister2`

The `create` routine may modify the value to be set and those changes will be stored as the initial value of the property. If the `create` routine returns a negative value, the new property value is not copied into the property and the create routine returns an error value.

The `set` routine is called before a new value is copied into the property. The `H5P_prp_set_func_t` callback function is defined as follows:

typedef herr_t (*H5P_prp_set_func_t)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`new_value`); The parameters to this callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being modified
const char *`name`	IN: The name of the property being modified
size_t `size`	IN: The size of the property in bytes
void **`new_value`	IN/OUT: Pointer to new value pointer for the property being modified

The `set` routine may modify the value pointer to be set and those changes will be used when setting the property's value. If the `set` routine returns a negative value, the new property value is not copied into the property and the `set` routine returns an error value. The `set` routine will not be called for the initial value, only the `create` routine will be called.

Note: The `set` callback function may be useful to range check the value being set for the property or may perform some transformation or translation of the value set. The `get` callback would then reverse the transformation or translation. A single `get` or `set` callback could handle multiple properties by performing different actions based on the property name or other properties in the property list.

The `get` routine is called when a value is retrieved from a property value. The `H5P_prp_get_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_get_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being queried
const char * `name`	IN: The name of the property being queried
size_t `size`	IN: The size of the property in bytes
void * `value`	IN/OUT: The value of the property being returned

The `get` routine may modify the value to be returned from the query and those changes will be returned to the calling routine. If the `set` routine returns a negative value, the query routine returns an error value.

The `delete` routine is called when a property is being deleted from a property list. The `H5P_prp_delete_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_delete_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list the property is being deleted from
const char * `name`	IN: The name of the property in the list
size_t `size`	IN: The size of the property in bytes
void * `value`	IN: The value for the property being deleted

The `delete` routine may modify the value passed in, but the value is not used by the library when the `delete` routine returns. If the `delete` routine returns a negative value, the property list delete routine returns an error value but the property is still deleted.

The `copy` routine is called when a new property list with this property is being created through a copy operation. The `H5P_prp_copy_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_copy_func_t`)(*const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

const char *`name`	IN: The name of the property being copied
size_t `size`	IN: The size of the property in bytes
void *`value`	IN/OUT: The value for the property being copied

The `copy` routine may modify the value to be set and those changes will be stored as the new value of the property. If the `copy` routine returns a negative value, the new property value is not copied into the property and the copy routine returns an error value.

The `compare` routine is called when a property list with this property is compared to another property list with the same property. The `H5P_prp_compare_func_t` callback function is defined as follows:

typedef int (*`H5P_prp_compare_func_t`)(*const void* *`value1`, *const void* *`value2`, *size_t* `size`); The parameters to the callback function are defined as follows:

const void *`value1`	IN: The value of the first property to compare
const void *`value2`	IN: The value of the second property to compare
size_t `size`	IN: The size of the property in bytes

The `compare` routine may *not* modify the values. The `compare` routine should return a positive value if `value1` is greater than `value2`, a negative value if `value2` is greater than `value1` and zero if `value1` and `value2` are equal.

The `close` routine is called when a property list with this property is being closed. The `H5P_prp_close_func_t` callback function is defined as follows:

typedef herr_t (*`H5P_prp_close_func_t`)(*hid_t* `prop_id`, *const char* *`name`, *size_t* `size`, *void* *`value`); The parameters to the callback function are defined as follows:

hid_t `prop_id`	IN: The identifier of the property list being closed
const char *`name`	IN: The name of the property in the list
size_t `size`	IN: The size of the property in bytes
void *`value`	IN: The value for the property being closed

The `close` routine may modify the value passed in, but the value is not used by the library when the `close` routine returns. If the `close` routine returns a negative value, the property list close routine returns an error value but the property list is still closed.

Parameters:

hid_t `class`	IN: Property list class to register permanent property within
const char * `name`	IN: Name of property to register
size_t `size`	IN: Size of property in bytes
void * `default`	IN: Default value for property in newly created property lists
H5P_prp_create_func_t `create`	IN: Callback routine called when a property list is being created and the property value will be initialized
H5P_prp_set_func_t `set`	IN: Callback routine called before a new value is copied into the property's value
H5P_prp_get_func_t `get`	IN: Callback routine called when a property value is retrieved from the property
H5P_prp_delete_func_t `delete`	IN: Callback routine called when a property is deleted from a property list
H5P_prp_copy_func_t `copy`	IN: Callback routine called when a property is copied from a property list

H5P_prp_compare_func_t `compare`	IN: Callback routine called when a property is compared with another property list
H5P_prp_close_func_t `close`	IN: Callback routine called when a property list is being closed and the property value will be disposed of

Returns:

Success: a non-negative value
Failure: a negative value

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Premove

Signature:

herr_t `H5Premove`(*hid_t* `plid`; *const char* *`name`)

Purpose:

Removes a property from a property list.

Description:

`H5Premove` removes a property from a property list.

Both properties which were in existence when the property list was created (i.e. properties registered with `H5Pregister`) and properties added to the list after it was created (i.e. added with `H5Pinsert1`) may be removed from a property list. Properties do not need to be removed from a property list before the list itself is closed; they will be released automatically when `H5Pclose` is called.

If a `close` callback exists for the removed property, it will be called before the property is released.

Parameters:

hid_t `plid`	IN: Identifier of the property list to modify
const char *`name`	IN: Name of property to remove

Returns:

Success: a non-negative value
Failure: a negative value

Fortran90 Interface: h5premove_f

```
SUBROUTINE h5premove_f(plid, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plid   ! Property list identifier
  CHARACTER(LEN=*), INTENT(IN) :: name ! Name of property to remove
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5premove_f
```

Name: H5Premove_filter

Signature:

herr_t `H5Premove_filter`(*hid_t* `plist`, *H5Z_filter_t* `filter`)

Purpose:

Delete one or more filters in the filter pipeline.

Description:

`H5Premove_filter` removes the specified `filter` from the filter pipeline in the dataset creation property list `plist`.

The `filter` parameter specifies the filter to be removed. Valid values for use in `filter` are as follows:

`H5Z_FILTER_ALL`	Removes all filters from the permanent filter pipeline.
`H5Z_FILTER_DEFLATE`	Data compression filter, employing the gzip algorithm
`H5Z_FILTER_SHUFFLE`	Data shuffling filter
`H5Z_FILTER_FLETCHER32`	Error detection filter, employing the Fletcher32 checksum algorithm
`H5Z_FILTER_SZIP`	Data compression filter, employing the SZIP algorithm

Additionally, user-defined filters can be removed with this routine by passing the filter identifier with which they were registered with the HDF5 Library.

Attempting to remove a filter that is not in the permanent filter pipeline is an error.

Note:

This function currently supports only the permanent filter pipeline; `plist` must be a dataset creation property list.

Parameters:

hid_t `plist_id`

IN: Dataset creation property list identifier.

H5Z_filter_t `filter`

IN: Filter to be deleted.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5premove_filter_f

```
SUBROUTINE h5premove_filter_f(prp_id, filter, hdferr)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Dataset creation property
                                       ! list identifier
  INTEGER, INTENT(IN) :: filter        ! Filter to be removed
                                       ! Valid values are:
                                       !     H5Z_FILTER_ALL_F
                                       !     H5Z_FILTER_DEFLATE_F
                                       !     H5Z_FILTER_SHUFFLE_F
                                       !     H5Z_FILTER_FLETCHER32_F
                                       !     H5Z_FILTER_SZIP_F
                                       !
```

```
    INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                         ! 0 on success, -1 on failure
  END SUBROUTINE h5premove_filter_f
```

History:

Release	**C**
1.6.3	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Pset

Signature:

herr_t `H5Pset`(*hid_t* `plid`, *const char* *`name`, *void* *`value`))

Purpose:

Sets a property list value.

Description:

`H5Pset` sets a new value for a property in a property list. If there is a `set` callback routine registered for this property, the `value` will be passed to that routine and any changes to the `value` will be used when setting the property value. The information pointed to by the `value` pointer (possibly modified by the `set` callback) is copied into the property list value and may be changed by the application making the `H5Pset` call without affecting the property value.

The property name must exist or this routine will fail.

If the `set` callback routine returns an error, the property value will not be modified.

This routine may not be called for zero-sized properties and will return an error in that case.

Parameters:

hid_t `plid`;	IN: Property list identifier to modify
const char *`name;`	IN: Name of property to modify
void *`value`;	IN: Pointer to value to set the property to

Returns:

Success: a non-negative value
Failure: a negative value

Fortran90 Interface: h5pset_f

```
SUBROUTINE h5pset_f(plid, name, value, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plid    ! Property list identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of property to set
  TYPE,   INTENT(IN) :: value           ! Property value
                                        ! Supported types are:
                                        !    INTEGER
                                        !    REAL
                                        !    DOUBLE PRECISION
                                        !    CHARACTER(LEN=*)
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_f
```

Name: H5Pset_alignment
Signature:

herr_t `H5Pset_alignment`(*hid_t* `plist`, *hsize_t* `threshold`, *hsize_t* `alignment`)

Purpose:

Sets alignment properties of a file access property list.

Description:

`H5Pset_alignment` sets the alignment properties of a file access property list so that any file object greater than or equal in size to `threshold` bytes will be aligned on an address which is a multiple of `alignment`. The addresses are relative to the end of the user block; the alignment is calculated by subtracting the user block size from the absolute file address and then adjusting the address to be a multiple of `alignment`.

Default values for `threshold` and `alignment` are one, implying no alignment. Generally the default values will result in the best performance for single-process access to the file. For MPI IO and other parallel systems, choose an alignment which is a multiple of the disk block size.

Parameters:

hid_t `plist`	IN: Identifier for a file access property list.
hsize_t `threshold`	IN: Threshold value. Note that setting the threshold value to 0 (zero) has the effect of a special case, forcing everything to be aligned.
hsize_t `alignment`	IN: Alignment value.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_alignment_f

```
SUBROUTINE h5pset_alignment_f(prp_id, threshold,  alignment, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id       ! Property list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: threshold  ! Threshold value
  INTEGER(HSIZE_T), INTENT(IN) :: alignment  ! Alignment value
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5pset_alignment_f
```

Name: H5Pset_alloc_time

Signature:

herr_t `H5Pset_alloc_time`(*hid_t* `plist_id`, *H5D_alloc_time_t* `alloc_time`)

Purpose:

Sets the timing for storage space allocation.

Description:

`H5Pset_alloc_time` sets up the timing for the allocation of storage space for a dataset's raw data. This property is set in the dataset creation property list `plist_id`.

Timing is specified in `alloc_time` with one of the following values:

`H5D_ALLOC_TIME_DEFAULT`	Allocate dataset storage space at the default time. (Defaults differ by storage method.)
`H5D_ALLOC_TIME_EARLY`	Allocate all space when the dataset is created. (Default for compact datasets.)
`H5D_ALLOC_TIME_INCR`	Allocate space incrementally, as data is written to the dataset. (Default for chunked storage datasets.) ♦ Chunked datasets: Storage space allocation for each chunk is deferred until data is written to the chunk. ♦ Contiguous datasets: Incremental storage space allocation for contiguous data is treated as late allocation. ♦ Compact datasets: Incremental allocation is not allowed with compact datasets; `H5Pset_alloc_time` will return an error.
`H5D_ALLOC_TIME_LATE`	Allocate all space when data is first written to the dataset. (Default for contiguous datasets.)

Note:

`H5Pset_alloc_time` is designed to work in concert with the dataset fill value and fill value write time properties, set with the functions `H5Pset_fill_value` and `H5Pset_fill_time`.

See H5Dcreate for further cross-references.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
H5D_alloc_time_t `alloc_time`	IN: When to allocate dataset storage space.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_alloc_time_f

```
SUBROUTINE h5pset_alloc_time_f(plist_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id  ! Dataset creation property
                                          ! list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: flag    ! Allocation time flag
                                          ! Possible values are:
                                          !    H5D_ALLOC_TIME_ERROR_F
                                          !    H5D_ALLOC_TIME_DEFAULT_F
                                          !    H5D_ALLOC_TIME_EARLY_F
                                          !    H5D_ALLOC_TIME_LATE_F
                                          !    H5D_ALLOC_TIME_INCR_F
```

```
          INTEGER, INTENT(OUT)       :: hdferr    ! Error code
                                                  ! 0 on success and -1 on failure
        END SUBROUTINE h5pset_alloc_time_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_attr_creation_order

Signature:

herr_t `H5Pset_attr_creation_order(` *hid_t* `ocpl_id`, *unsigned* `crt_order_flags`)

Purpose:

Sets tracking and indexing of attribute creation order.

Description:

`H5Pset_attr_creation_order` sets flags specifying whether to track and index attribute creation order on an object.

`ocpl_id` is a dataset or group creation property list identifier. The term `ocpl`, for object creation property list, is used when different types of objects may be involved.

`crt_order_flags` contains flags with the following meanings:

`H5P_CRT_ORDER_TRACKED`	Attribute creation order is tracked but not necessarily indexed.
`H5P_CRT_ORDER_INDEXED`	Attribute creation order is indexed (requires `H5P_CRT_ORDER_TRACKED`).

Default behavior is that attribute creation order is neither tracked nor indexed.

Parameters:

hid_t `ocpl_id`	IN: Object creation property list identifier
unsigned `crt_order_flags`	IN: Flags specifying whether to track and index attribute creation order *Default: No flag set; attribute creation order is neither tracked not indexed.*

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pget_attr_creation_order_f

```
SUBROUTINE h5pget_attr_creation_order_f(ocpl_id, crt_order_flags, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: ocpl_id
                                ! Object (group or dataset) creation property
                                ! list identifier
  INTEGER, INTENT(OUT) :: crt_order_flags
                                ! Flags specifying whether to track
                                ! and index attribute creation order
  INTEGER, INTENT(OUT) :: hdferr ! Error code
                                ! 0 on success and -1 on failure
END SUBROUTINE h5pget_attr_creation_order_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_attr_phase_change

Signature:

herr_t `H5Pset_attr_phase_change`(*hid_t* `ocpl_id`, *unsigned* `max_compact`, *unsigned* `min_dense`)

Purpose:

Sets attribute storage phase change thresholds.

Description:

`H5Pset_attr_phase_change` sets threshold values for attribute storage on an object. These thresholds determine the point at which attribute storage changes from compact storage (i.e., storage in the object header) to dense storage (i.e., storage in a heap and indexed with a B-tree).

In the general case, attributes are initially kept in compact storage. When the number of attributes exceeds `max_compact`, attribute storage switches to dense storage. If the number of attributes subsequently falls below `min_dense`, the attributes are returned to compact storage.

If `max_compact` is set to `0` (zero), dense storage always used.

`ocpl_id` is a dataset or group creation property list identifier. The term `ocpl`, for object creation property list, is used when different types of objects may be involved.

Parameters:

hid_t `ocpl_id`	IN: Object (group or dataset) creation property list identifier
unsigned `max_compact`	IN: Maximum number of attributes to be stored in compact storage *(Default: 8)*
unsigned `min_dense`	IN: Minimum number of attributes to be stored in dense storage *(Default: 6)*

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_attr_phase_change_f

```
SUBROUTINE h5pset_attr_phase_change_f(ocpl_id, max_compact, min_dense, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: ocpl_id
                               ! Object (dataset or group) creation property
                               ! list identifier
  INTEGER, INTENT(IN) :: max_compact
                               ! Maximum number of attributes to be stored in
                               ! compact storage (Default: 8)
  INTEGER, INTENT(IN) :: min_dense
                               ! Minimum number of attributes to be stored in
                               ! dense storage (Default: 6)
  INTEGER, INTENT(OUT) :: hdferr
                               ! Error code
                               ! 0 on success and -1 on failure
END SUBROUTINE h5pset_attr_phase_change_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_btree_ratios

Signature:

herr_t `H5Pset_btree_ratios`(*hid_t* `plist`, *double* `left`, *double* `middle`, *double* `right`)

Purpose:

Sets B-tree split ratios for a dataset transfer property list.

Description:

`H5Pset_btree_ratios` sets the B-tree split ratios for a dataset transfer property list. The split ratios determine what percent of children go in the first node when a node splits.

The ratio `left` is used when the splitting node is the left-most node at its level in the tree; the ratio `right` is used when the splitting node is the right-most node at its level; and the ratio `middle` is used for all other cases.

A node which is the only node at its level in the tree uses the ratio `right` when it splits.

All ratios are real numbers between 0 and 1, inclusive.

Parameters:

hid_t `plist`	IN: The dataset transfer property list identifier.
double `left`	IN: The B-tree split ratio for left-most nodes.
double `right`	IN: The B-tree split ratio for right-most nodes and lone nodes.
double `middle`	IN: The B-tree split ratio for all other nodes.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_btree_ratios_f

```
SUBROUTINE h5pset_btree_ratios_f(prp_id, left, middle, right, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id
                                   ! Property list identifier
  REAL, INTENT(IN) :: left         ! The B-tree split ratio for left-most nodes
  REAL, INTENT(IN) :: middle       ! The B-tree split ratio for all other nodes
  REAL, INTENT(IN) :: right        ! The B-tree split ratio for right-most
                                   ! nodes and lone nodes.
  INTEGER, INTENT(OUT) :: hdferr   ! Error code
                                   ! 0 on success and -1 on failure
END SUBROUTINE h5pset_btree_ratios_f
```

Name: H5Pset_buffer

Signature:

herr_t `H5Pset_buffer`(*hid_t* `plist`, *hsize_t* `size`, *void* `*tconv`, *void* `*bkg`)

Purpose:

Sets type conversion and background buffers.

Description:

Given a dataset transfer property list, `H5Pset_buffer` sets the maximum size for the type conversion buffer and background buffer and optionally supplies pointers to application-allocated buffers. If the buffer size is smaller than the entire amount of data being transferred between the application and the file, and a type conversion buffer or background buffer is required, then strip mining will be used.

Note that there are minimum size requirements for the buffer. Strip mining can only break the data up along the first dimension, so the buffer must be large enough to accommodate a complete slice that encompasses all of the remaining dimensions. For example, when strip mining a 100x200x300 hyperslab of a simple data space, the buffer must be large enough to hold 1x200x300 data elements. When strip mining a 100x200x300x150 hyperslab of a simple data space, the buffer must be large enough to hold 1x200x300x150 data elements.

If `tconv` and/or `bkg` are null pointers, then buffers will be allocated and freed during the data transfer.

The default value for the maximum buffer is 1 Mb.

Parameters:

hid_t `plist`	IN: Identifier for the dataset transfer property list.
hsize_t `size`	IN: Size, in bytes, of the type conversion and background buffers.
void `tconv`	IN: Pointer to application-allocated type conversion buffer.
void `bkg`	IN: Pointer to application-allocated background buffer.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_buffer_f

```
SUBROUTINE h5pset_buffer_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: plist_id ! Dataset transfer property
                                           ! list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: size     ! Conversion buffer size
  INTEGER, INTENT(OUT)         :: hdferr   ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pset_buffer_f
```

History:

Release	**C**
1.6.0	The `size` parameter has changed from type *hsize_t* to *size_t*.
1.4.0	The `size` parameter has changed to type *hsize_t*.

Last modified: 15 April 2009

Name: H5Pset_cache

Signature:

herr_t `H5Pset_cache(` *hid_t* `plist_id`, *int* `mdc_nelmts`, *size_t* `rdcc_nelmts`, *size_t* `rdcc_nbytes`, *double* `rdcc_w0`)

Purpose:

Sets the raw data chunk cache parameters.

Description:

`H5Pset_cache` sets the number of elements, the total number of bytes, and the preemption policy value in the raw data chunk cache.

The *plist_id* is a file access property list.

The number of elements (objects) in the raw data chunk cache is *rdcc_nelmts*. The total size of the raw data chunk cache and the preemption policy are *rdcc_nbytes* and *rdcc_w0*, respectively.

Any (or all) of the `H5Pget_cache` pointer arguments may be null pointers.

The *rdcc_w0* value should be between 0 and 1 inclusive and indicates how much chunks that have been fully read are favored for preemption. A value of zero means fully read chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of one means fully read chunks are always preempted before other chunks.

The `*mdc_nelmts` parameter is no longer used; any value passed in that parameter is ignored.

Note:

Raw dataset chunk caching is not currently supported when using the MPI I/O and MPI POSIX file drivers in read/write mode; see `H5Pset_fapl_mpio` and `H5Pset_fapl_mpiposix`, respectively. When using one of these file drivers, all calls to `H5Dread` and `H5Dwrite` will access the disk directly, and `H5Pset_cache` will have no effect on performance.

Raw dataset chunk caching is supported when these drivers are used in read-only mode.

Parameters:

hid_t `plist_id`	IN: Identifier of the file access property list.
int `mdc_nelmts`	IN: *No longer used; any value passed is ignored.*
size_t `rdcc_nelmts`	IN: Number of elements (objects) in the raw data chunk cache.
size_t `rdcc_nbytes`	IN: Total size of the raw data chunk cache, in bytes.
double `rdcc_w0`	IN: Preemption policy.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_cache_f

```
SUBROUTINE h5pset_cache_f(prp_id, mdc_nelmts,rdcc_nelmts, rdcc_nbytes, rdcc_w0, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id         ! Property list identifier
  INTEGER, INTENT(IN) :: mdc_nelmts            ! Number of elements (objects)
                                               ! in the meta data cache
  INTEGER(SIZE_T), INTENT(IN) :: rdcc_nelmts  ! Number of elements (objects)
                                               ! in the meta data cache
  INTEGER(SIZE_T), INTENT(IN) :: rdcc_nbytes  ! Total size of the raw data
                                               ! chunk cache, in bytes
  REAL, INTENT(IN) :: rdcc_w0                  ! Preemption policy
```

```
          INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                                      ! 0 on success and -1 on failure
        END SUBROUTINE h5pset_cache_f
```

History:

Release	**Change**
1.8.0	In C, use of the `mdc_nelmts` parameter discontinued. Metadata cache configuration is managed with `H5Pset_mdc_config` and `H5Pget_mdc_config`.
1.6.1	Fortran `rdcc_nbytes` parameter type changed to `INTEGER(SIZE_T)`.
1.6.0	In C, the `rdcc_nbytes` and `rdcc_nelmts` parameters changed from type *int* to *size_t*.

Name: H5Pset_char_encoding

Signature:

herr_t `H5Pset_char_encoding`(*hid_t* `plist_id`, *H5T_cset_t* `encoding`)

Purpose:

Sets the character encoding used to encode a string.

Description:

`H5Pset_char_encoding` sets the character encoding used to encode strings or object names that are created with the property list `plist_id`.

Valid values for `encoding` are defined in `H5Tpublic.h` and include the following:

`H5T_CSET_ASCII`	US ASCII
`H5T_CSET_UTF8`	UTF-8 Unicode encoding

Parameters:

hid_t `plist_id`	IN: Property list identifier
H5T_cset_t `encoding`	IN: String encoding character set

Returns:

Returns a non-negative valule if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_char_encoding_f

```
SUBROUTINE h5pset_char_encoding_f(plist_id, encoding, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id
                                   ! Property list identifier
  INTEGER, INTENT(IN) :: encoding ! String encoding character set:
                                   !   H5T_CSET_ASCII_F -> US ASCII
                                   !   H5T_CSET_UTF8_F -> UTF-8 Unicode encoding
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
                                   ! 0 on success and -1 on failure
END SUBROUTINE h5pset_char_encoding_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_chunk

Signature:

herr_t `H5Pset_chunk`(*hid_t* `plist`, *int* `ndims`, *const hsize_t* * `dim`)

Purpose:

Sets the size of the chunks used to store a chunked layout dataset.

Description:

`H5Pset_chunk` sets the size of the chunks used to store a chunked layout dataset. This function is only valid for dataset creation property lists.

The `ndims` parameter currently must be the same size as the rank of the dataset.

The values of the `dim` array define the size of the chunks to store the dataset's raw data. The unit of measure for `dim` values is *dataset elements*.

As a side-effect of this function, the layout of the dataset is changed to `H5D_CHUNKED`, if it is not already so set. (See `H5Pset_layout`.)

Note regarding fixed-size datasets:
Chunk size cannot exceed the size of a fixed-size dataset. For example, a dataset consisting of a 5x4 fixed-size array cannot be defined with 10x10 chunks.

Parameters:

hid_t `plist`	IN: Dataset creation property list identifier.
int `ndims`	IN: The number of dimensions of each chunk.
const hsize_t * `dim`	IN: An array defining the size, in dataset elements, of each chunk.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_chunk_f

```
SUBROUTINE h5pset_chunk_f(prp_id, ndims, dims, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(IN) :: ndims          ! Number of chunk dimensions
  INTEGER(HSIZE_T), DIMENSION(ndims), INTENT(IN) :: dims
                                        ! Array containing sizes of
                                        ! chunk dimensions
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_chunk_f
```

Last modified: 8 May 2009

Name: H5Pset_chunk_cache

Signature:

herr_t `H5Pset_chunk_cache`(*hid_t* `dapl_id`, *size_t* `rdcc_nslots`, *size_t* `rdcc_nbytes`, *double* `rdcc_w0`)

Purpose:

Sets the raw data chunk cache parameters.

Motivation:

`H5Pset_chunk_cache` is used to adjust the chunk cache parameters on a per-dataset basis, as opposed to a global setting for the file. The optimum chunk cache parameters vary wildly with different data layout and access patterns, so for optimal performance they must be set individually for each dataset. It may also be beneficial to reduce the size of the chunk cache for datasets whose performance is not important in order to save memory space.

Description:

`H5Pset_chunk_cache` sets the number of elements, the total number of bytes, and the preemption policy value in the raw data chunk cache on a dataset access property list. After calling this function, the values set in the property list will override the values in the file's file access property list.

The raw data chunk cache inserts chunks into the cache by first computing a hash value using the address of a chunk, then using that hash value as the chunk's index into the table of cached chunks. The size of this hash table, i.e., and the number of possible hash values, is determined by the `rdcc_nslots` parameter. If a different chunk in the cache has the same hash value, this causes a collision, which reduces efficiency. If inserting the chunk into cache would cause the cache to be too big, then the cache is pruned according to the `rdcc_w0` parameter.

Parameters:

hid_t `dapl_id`	IN: Dataset access property list identifier.
size_t `rdcc_nslots`	IN:The number of chunk slots in the raw data chunk cache for this dataset. Increasing this value reduces the number of cache collisions, but slightly increases the memory used. Due to the hashing strategy, this value should ideally be a prime number. As a rule of thumb, this value should be at least 10 times the number of chunks that can fit in `rdcc_nbytes` bytes. For maximum performance, this value should be set approximately 100 times that number of chunks. The default value is `521`. If the value passed is `H5D_CHUNK_CACHE_NSLOTS_DEFAULT`, then the property will not be set on `dapl_id` and the parameter will come from the file access property list used to open the file.
size_t `rdcc_nbytes`	IN: The total size of the raw data chunk cache for this dataset. In most cases increasing this number will improve performance, as long as you have enough free memory. The default size is 1 MB. If the value passed is `H5D_CHUNK_CACHE_NBYTES_DEFAULT`, then the property will not be set on `dapl_id` and the parameter will come from the file access property list.
double `rdcc_w0`	IN: The chunk preemption policy for this dataset. This must be between `0` and `1` inclusive and indicates the weighting according to which chunks which have been fully read or written are penalized when determining which chunks

to flush from cache. A value of `0` means fully read or written chunks are treated no differently than other chunks (the preemption is strictly LRU) while a value of `1` means fully read or written chunks are always preempted before other chunks. If your application only reads or writes data once, this can be safely set to `1`. Otherwise, this should be set lower, depending on how often you re-read or re-write the same data.

The default value is `0.75`. If the value passed is `H5D_CHUNK_CACHE_W0_DEFAULT`, then the property will not be set on `dapl_id` and the parameter will come from the file access property list.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The following code sets the chunk cache to use a hash table with 12421 elements and a maximum size of 16 MB, while using the preemption policy specified for the entire file:

```
H5Pset_chunk_cache(dapl_id, 12421, 16*1024*1024, H5D_CHUNK_CACHE_W0_DEFAULT);
```

See Also:

`H5Pget_chunk_cache`, `H5Pset_cache`

History:

Release	Change
1.8.3	C function introduced in this release.

Name: H5Pset_copy_object

Signature:

herr_t `H5Pset_copy_object`(*hid_t* `ocp_plist_id`, *unsigned* `copy_options`)

Purpose:

Sets properties to be used when an object is copied.

Description:

`H5Pset_copy_object` sets properties in the object copy property list `ocp_plist_id` that will be invoked when a new copy is made of an existing object.

`ocp_plist_id` is the object copy property list and specifies the properties governing the copying of the object.

Several flags, described in the following table, are available for inclusion in the object copy property list:

`H5O_COPY_SHALLOW_HIERARCHY_FLAG`	Copy only immediate members of a group. *Default behavior, without flag:* Recursively copy all objects below the group.
`H5O_COPY_EXPAND_SOFT_LINK_FLAG`	Expand soft links into new objects. *Default behavior, without flag:* Keep soft links as they are.
`H5O_COPY_EXPAND_EXT_LINK_FLAG`	Expand external link into new objects. *Default behavior, without flag:* Keep external links as they are.
`H5O_COPY_EXPAND_REFERENCE_FLAG`	Copy objects that are pointed to by references. *Default behavior, without flag:* Update only the values of object references.
`H5O_COPY_WITHOUT_ATTR_FLAG`	Copy object without copying attributes. *Default behavior, without flag:* Copy object along with all its attributes.

Parameters:

hid_t `ocp_plist_id`	Object copy property list identifier
unsigned `copy_options`	Copy option(s) to be set

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_copy_object_f

```
SUBROUTINE h5pset_copy_object_f(ocp_plist_id, copy_options, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: ocp_plist_id
                              ! Object copy property list identifier
  INTEGER, INTENT(IN) :: copy_options
                              ! Copy option(s) to be set, valid options are:
                              !   H5O_COPY_SHALLOW_HIERARCHY_F
                              !   H5O_COPY_EXPAND_SOFT_LINK_F
                              !   H5O_COPY_EXPAND_EXT_LINK_F
                              !   H5O_COPY_EXPAND_REFERENCE_F
                              !   H5O_COPY_WITHOUT_ATTR_FLAG_F
  INTEGER, INTENT(OUT) :: hdferr
                              ! Error code
                              ! 0 on success and -1 on failure
END SUBROUTINE h5pset_copy_object_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_create_intermediate_group

Signature:

herr_t `H5Pset_create_intermediate_group`(*hid_t* `lcpl_id`, *unsigned* `crt_intermed_group`)

Purpose:

Specifies in property list whether to create missing intermediate groups.

Description:

`H5Pset_create_intermediate_group` specifies whether to set the link creation property list `lcpl_id` so that calls to functions that create objects in groups different from the current working group will create intermediate groups that may be missing in the path of a new or moved object.

Functions that create objects in or move objects to a group other than the current working group make use of this property. `H5Gcreate_anon` and `H5Lmove` are examles of such functions.

If `crt_intermed_group` is positive, the `H5G_CRT_INTMD_GROUP` will be added to `lcpl_id` (if it is not already there). Missing intermediate groups will be created upon calls to functions such as those listed above that use `lcpl_id`.

If `crt_intermed_group` is non-positive, the `H5G_CRT_INTMD_GROUP`, if present, will be removed from `lcpl_id`. Missing intermediate groups will *not* be created upon calls to functions such as those listed above that use `lcpl_id`.

Parameters:

hid_t `lcpl_id`	Link creation property list identifier
unsigned `crt_intermed_group`	Flag specifying whether to create intermediate groups upon the creation of an object

Returns:

Returns a non-negative valule if successful; otherwise returns a negative value.

Example:

The following call sets the link creation property list `lcpl_id` such that a call to `H5Gcreate_anon` or other function using `lcpl_id` will create any missing groups in the path to the new object:

```
herr_t ret_value = H5Pset_create_intermediate_group(lcpl_id, 1)
```

Fortran90 Interface:

```
SUBROUTINE h5pset_create_inter_group_f(lcpl_id, crt_intermed_group, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: lcpl_id
                           ! Link creation property list identifier
  INTEGER, INTENT(IN) :: crt_intermed_group
                           ! Specifying whether to create intermediate groups
                           ! upon the creation of an object
  INTEGER, INTENT(OUT) :: hdferr
                           ! Error code
                           ! 0 on success and -1 on failure
END SUBROUTINE h5pset_create_inter_group_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_data_transform

Signature:

herr_t `H5Pset_data_transform` (*hid_t* `plist_id`, *const char* *`expression`)

Purpose:

Sets a data transform expression.

Description:

`H5Pset_data_transform` sets the data transform to be used for reading and writing data. This function operates on the dataset transfer property lists plist_id.

The expression parameter is a string containing an algebraic expression, such as `(5/9.0)*(x-32)` or `x*(x-5)`. When a dataset is read or written with this property list, the transform expression is applied with the `x` being replaced by the values in the dataset. When reading data, the values in the file are not changed and the transformed data is returned to the user.

Data transforms can only be applied to integer or floating-point datasets. Order of operations is obeyed and the only supported operations are +, -, *, and /. Parentheses can be nested arbitrarily and can be used to change precedence.

When writing data back to the dataset, the transformed data is written to the file and there is no way to recover the original values to which the transform was applied.

Parameters:

hid_t `plist_id`	IN: Identifier of the property list or class
const char *`expression`	IN: Pointer to the null-terminated data transform expression

Returns:

Success: a non-negative value
Failure: a negative value

Fortran90 Interface: SUBROUTINE h5pset_data_transform_f

```
SUBROUTINE h5pset_data_transform_f(plist_id, expression, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id
                                    ! Identifier of the property list or class
  CHARACTER(LEN=*), INTENT(IN) :: expression
                                    ! Buffer to hold transform expression
  INTEGER, INTENT(OUT) :: hdferr    ! Error code
                                    ! 0 on success and -1 on failure
END SUBROUTINE h5pset_data_transform_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_deflate

Signature:

herr_t `H5Pset_deflate`(*hid_t* `plist`, *uint* `level`)

Purpose:

Sets compression method and compression level.

Description:

`H5Pset_deflate` sets the compression method for a dataset creation property list to `H5D_COMPRESS_DEFLATE` and the compression level to `level`, which should be a value from zero to nine, inclusive.

Lower compression levels are faster but result in less compression.

This is the same algorithm as used by the GNU gzip program.

Parameters:

hid_t `plist`	IN: Identifier for the dataset creation property list.
uint `level`	IN: Compression level.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_deflate_f

```
SUBROUTINE h5pset_deflate_f(prp_id, level, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(IN)        :: level  ! Compression level
  INTEGER, INTENT(OUT)       :: hdferr ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pset_deflate_f
```

Name: H5Pset_driver

Signature:

herr_t `H5Pset_driver(` *hid_t* `plist_id`, *hid_t* `new_driver_id`, *const void* `*new_driver_info`)

Purpose:

Sets a file driver.

Description:

`H5Pset_driver` sets the file driver, `new_driver_id`, for a file access or data transfer property list, `plist_id`, and supplies an optional struct containing the driver-specific properties, `new_driver_info`.

The driver properties will be copied into the property list and the reference count on the driver will be incremented, allowing the caller to close the driver identifier but still use the property list.

Note:

`H5Pset_driver` and `H5Pget_driver_info` are used only when creating a virtual file driver (VFD) in the virtual file layer (VFL). For further information, see “Virtual File Layer” and “List of VFL Functions” in the *HDF5 Technical Notes*.

Parameters:

hid_t `plist_id`

IN: File access or data transfer property list identifier.

hid_t `new_driver_id`

IN: Driver identifier.

const void * `new_driver_info`

IN: Optional struct containing driver properties.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.2	Function publicized in this release; previous releases described this function only in the virtual file driver documentation.

Name: H5Pset_dxpl_mpio

Signature:

herr_t `H5Pset_dxpl_mpio`(*hid_t* `dxpl_id`, *H5FD_mpio_xfer_t* `xfer_mode`)

Purpose:

Sets data transfer mode.

Description:

`H5Pset_dxpl_mpio` sets the data transfer property list `dxpl_id` to use transfer mode `xfer_mode`. The property list can then be used to control the I/O transfer mode during data I/O operations.

Valid transfer modes are as follows:

`H5FD_MPIO_INDEPENDENT`
: Use independent I/O access (default).

`H5FD_MPIO_COLLECTIVE`
: Use collective I/O access.

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier.
H5FD_mpio_xfer_t `xfer_mode`	IN: Transfer mode.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE h5pset_dxpl_mpio_f(prp_id, data_xfer_mode, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(IN) :: data_xfer_mode ! Data transfer mode
                                        ! Possible values are:
                                        !    H5FD_MPIO_INDEPENDENT_F
                                        !    H5FD_MPIO_COLLECTIVE_F
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_dxpl_mpio_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pset_dxpl_mpio_chunk_opt

Signature:

herr_t `H5Pset_dxpl_mpio_chunk_opt` (*hid_t* `dxpl_id`, *H5FD_mpio_chunk_opt_t* `opt_mode`)

Purpose:

Sets a flag specifying linked-chunk I/O or multi-chunk I/O.

Description:

`H5Pset_dxpl_mpio_chunk_opt` specifies whether I/O is to be performed as linked-chunk I/O or as multi-chunk I/O. This function overrides the HDF5 Library's internal algorithm for determining which mechanism to use.

When an application uses collective I/O with chunked storage, the HDF5 Library normally uses an internal algorithm to determine whether that I/O activity should be conducted as one linked-chunk I/O or as multi-chunk I/O. `H5Pset_dxpl_mpio_chunk_opt` is provided so that an application can override the library's alogorithm in circumstances where the library might lack the information needed to make an optimal desision.

`H5Pset_dxpl_mpio_chunk_opt` works by setting one of the following flags in the parameter `opt_mode`:

`H5FD_MPIO_CHUNK_ONE_IO`	Do one link chunked I/O.
`H5FD_MPIO_CHUNK_MULTI_IO`	Do multi-chunked I/O.

This function works by setting a corresponding property in the dataset transfer property list `dxpl_id`.

The library perform I/O in the specified manner *unless* it determines that the low-level MPI IO package does not support the requested behavior; in such cases, the HDF5 Library will internally use independent I/O.

Use of this function is optional.

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
H5FD_mpio_chunk_opt_t `opt_mode`	IN: Optimization flag specifying linked-chunk I/O or multi-chunk I/O

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Name: H5Pset_dxpl_mpio_chunk_opt_num

Signature:

herr_t `H5Pset_dxpl_mpio_chunk_opt_num` (*hid_t* `dxpl_id`, *unsigned* `num_chunk_per_proc`)

Purpose:

Sets a numeric threshold for linked-chunk I/O.

Description:

`H5Pset_dxpl_mpio_chunk_opt_num` sets a numeric threshold for the use of linked-chunk I/O.

The library will calculate the average number of chunks selected by each process when doing collective access with chunked storage. If the number is greater than the threshold set in `num_chunk_per_proc`, the library will use linked-chunk I/O; otherwise, a separate I/O process will be invoked for each chunk (multi-chunk I/O).

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
unsigned `num_proc_per_chunk`	IN: Numeric threshold for performing linked-chunk I/O

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Name: H5Pset_dxpl_mpio_chunk_opt_ratio

Signature:

herr_t `H5Pset_dxpl_mpio_chunk_opt_ratio` (*hid_t* `dxpl_id`, *unsigned* `percent_proc_per_chunk`)

Purpose:

Sets a ratio threshold for collective I/O.

Description:

`H5Pset_dxpl_mpio_chunk_opt_ratio` sets a threshold for the use of collective I/O based on the ratio of processes with collective access to a dataset with chunked storage. The decision whether to use collective I/O is made on a per-chunk basis.

The library will calculate the percentage of the total number of processes, the ratio, that hold selections in each chunk. If that percentage is greater than the threshold set in `percent_proc_per_chunk`, the library will do collective I/O for this chunk; otherwise, independent I/O will be done for the chunk.

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
unsigned `percent_proc_per_chunk`	IN: Percent threshold, on the number of processes holding selections per chunk, for performing linked-chunk I/O

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Name: H5Pset_dxpl_mpio_collective_opt

Signature:

herr_t `H5Pset_dxpl_mpio_collective_opt` (*hid_t* `dxpl_id`, *H5FD_mpio_collective_opt_t* `opt_mode`)

Purpose:

Sets a flag governing the use of independent versus collective I/O.

Description:

`H5Pset_dxpl_mpio_collective_opt` enables an application to specify that the HDF5 Library will use independent I/O internally when the dataset transfer property list `dxpl_id` is set for collective I/O, i.e., with `H5FD_MPIO_COLLECTIVE` specified. This allows the application greater control over low-level I/O while maintaining the collective interface at the application level.

`H5Pset_dxpl_mpio_collective_opt` works by setting one of the following flags in the parameter `opt_mode`:

`H5FD_MPIO_COLLECTIVE_IO`	Use collective I/O. *(Default)*
`H5FD_MPIO_INDIVIDUAL_IO`	Use independent I/O.

This function should be used only when `H5FD_MPIO_COLLECTIVE` has been set through `H5Pset_dxpl_mpio`. In such situations, normal behavior would be to use low-level collective I/O functions, but the library will use low-level MPI independent I/O functions when `H5FD_MPIO_INDIVIDUAL_IO` is set.

Use of this function is optional.

Parameters:

hid_t `dxpl_id`	IN: Data transfer property list identifier
H5FD_mpio_collective_opt_t `opt_mode`	IN: Optimization flag specifying the use of independent or collective I/O

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Name: H5Pset_dxpl_multi

Signature:

herr_t `H5Pset_dxpl_multi`(*hid_t* `dxpl_id`, *const hid_t* *`memb_dxpl`)

Purpose:

Sets the data transfer property list for the multi-file driver.

Description:

`H5Pset_dxpl_multi` sets the data transfer property list `dxpl_id` to use the multi-file driver for each memory usage type `memb_dxpl[]`.

`H5Pset_dxpl_multi` can only be used after the member map has been set with `H5Pset_fapl_multi`.

Parameters:

hid_t `dxpl_id`,	IN: Data transfer property list identifier.
const hid_t *`memb_dxpl`	IN: Array of data access property lists.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pset_edc_check

Signature:

herr_t `H5Pset_edc_check`(*hid_t* `plist`, *H5Z_EDC_t* `check`)

Purpose:

Sets whether to enable error-detection when reading a dataset.

Description:

`H5Pset_edc_check` sets the dataset transfer property list `plist` to enable or disable error detection when reading data.

Whether error detection is enabled or disabled is specified in the `check` parameter. Valid values are as follows:

`H5Z_ENABLE_EDC` (default)
`H5Z_DISABLE_EDC`

The error detection algorithm used is the algorithm previously specified in the corresponding dataset creation property list.

This function does not affect the use of error detection when writing data.

Note:

The initial error detection implementation, Fletcher32 checksum, supports error detection for chunked datasets only.

Parameters:

hid_t `plist`	IN: Dataset transfer property list identifier.
H5Z_EDC_t `check`	IN: Specifies whether error checking is enabled or disabled for dataset read operations.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_edc_check_f

```
SUBROUTINE h5pset_edc_check_f(prp_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id   ! Dataset transfer property
                                         ! list identifier
  INTEGER, INTENT(IN)        :: flag     ! EDC flag; possible values
                                         !    H5Z_DISABLE_EDC_F
                                         !    H5Z_ENABLE_EDC_F
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure

END SUBROUTINE h5pset_edc_check_f
```

History:

Release	C
1.6.0	Function introduced in this release.

Last modified: 8 May 2009

Name: H5Pset_elink_acc_flags

Signature:

herr_t `H5Pset_elink_acc_flags(` *hid_t* `lapl_id`, *unsigned* `flags`)

Purpose:

Sets the external link traversal file access flag in a link access property list.

Motivation:

`H5Pset_elink_acc_flags` is used to adjust the file access flag used to open files reached through external links. This may be useful to, for example, prevent modifying files accessed through an external link. Otherwise, the target file is opened with whatever flag was used to open the parent.

Description:

`H5Pset_elink_acc_flags` specifies the file access flag to use to open the target file of an external link. This allows read-only access of files reached through an external link in a file opened with write access, or vice-versa.

The library will normally use the file access flag used to open the parent file as the file access flag for the target file. This function provides a way to override that behaviour. The external link traversal callback function set by `H5Pset_elink_cb` can override the setting from `H5Pset_elink_acc_flags`.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier
unsigned `flags`	IN: The access flag for external link traversal. Valid values include:

`H5F_ACC_RDWR`	Causes files opened through external links to be opened with write access.
`H5F_ACC_RDONLY`	Causes files opened through external links to be opened with read-only access.
`H5F_ACC_DEFAULT`	Removes any external link file access flag setting from `lapl_id`, causing the file access flag setting to be taken from the parent file.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

The following code sets the link access property list `lapl_id` to open external link target files with read-only access:

```
status = H5Pset_elink_acc_flags(lapl_id, H5F_ACC_RDONLY);
```

See Also:

`H5Pget_elink_acc_flags`, `H5Pset_elink_cb`, `H5Fopen`, `H5Lcreate_external`

History:

Release	**Change**
1.8.3	C function introduced in this release.

Last modified: 8 May 2009

Name: H5Pset_elink_cb

Signature:

herr_t `H5Pset_elink_cb`(*hid_t* `lapl_id`, *H5L_elink_traverse_t* `func`, *void* *`op_data`)

Purpose:

Sets the external link traversal callback function in a link access property list.

Motivation:

`H5Pset_elink_cb` is used to add a callback function that is called when traversing an external link. The callback can then set access permissions, modify the file access property list, modify the parent or target file, or take any other user defined action.

Description:

`H5Pset_elink_cb` sets the user-defined external link traversal callback function in the link access property list lapl_id.

The callback function may adjust the file access property list and file access flag to use when opening a file through an external link. The callback will be made immediately before opening the target file.

The callback will be made after the file access property list set by `H5Pset_elink_fapl` and the file access flag set by `H5Pset_elink_acc_flags` are applied, so changes made by this callback function take precedence.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier.
H5L_elink_traverse_t `func`	IN: User-defined external link traversal callback function.
void *`op_data`	IN: User-defined input data for the callback function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Example Usage:

This example defines a callback function that prints the name of the target file every time an external link is followed, and sets this callback function on `lapl_id`

```
herr_t elink_callback(const char *parent_file_name, const char
        *parent_group_name, const char *child_file_name, const char
        *child_object_name, unsigned *acc_flags, hid_t fapl_id, void *op_data) {
    puts(child_file_name);
    return 0;
}

int main(void) {
    hid_t lapl_id = H5Pcreate(H5P_LINK_ACCESS);
    H5Pset_elink_cb(lapl_id, elink_callback, NULL);
    ...
}
```

See Also:

`H5Pget_elink_cb`, `H5Pset_elink_fapl`, `H5Pset_elink_acc_flags`, `H5Fopen`, `H5Lcreate_external`

`H5L_elink_traverse_t`

History:

Release	Change
1.8.3	C function introduced in this release.

Last modified: 2 April 2009

Name: H5Pset_elink_fapl

Signature:

herr_t `H5Pset_elink_fapl(` *hid_t* `lapl_id`, *hid_t* `fapl_id`)

Purpose:

Sets a file access property list for use in accessing a file pointed to by an external link.

Description:

`H5Pset_elink_fapl` sets the file access property list, `fapl_id`, to be used when accessing the target file of an external link associated with `lapl_id`.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier
hid_t `fapl_id`	IN: File access property list identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

See Also:

`H5Pget_elink_fapl`
`H5Lcreate_external`

Fortran90 Interface:

None.

History:

Release	**Change**
1.9.0	C function introduced in this release.

Name: H5Pset_elink_prefix

Signature:

herr_t `H5Pset_elink_prefix`(*hid_t* `lapl_id`, *const char* *`prefix`)

Purpose:

Sets prefix to be applied to external link paths.

Description:

`H5Pset_elink_prefix` sets the prefix to be applied to the path of any external links traversed. The prefix is prepended to the filename stored in the external link.

The prefix is specified in the user-allocated buffer `prefix` and set in the link access property list `lapl_id`. The buffer should not be freed until the property list has been closed.

Parameters:

hid_t `lapl_id`	IN: Link access property list identifier
const char *`prefix`	IN: Prefix to be applied to external link paths

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_est_link_info

Signature:

herr_t `H5Pset_est_link_info`(*hid_t* `gcpl_id`, *unsigned* `est_num_entries`, *unsigned* `est_name_len`)

Purpose:

Sets estimated number of links and length of link names in a group.

Description:

`H5Pset_est_link_info` inserts two settings into the group creation property list `gcpl_id`: the estimated number of links that are expected to be inserted into a group created with the property list and the estimated average length of those link names.

The estimated number of links is passed in `est_num_entries`.

The estimated average length of the anticipated link names is passed in `est_name_len`.

The values for these two settings are multiplied to compute the initial local heap size (for old-style groups, if the local heap size hint is not set) or the initial object header size for (new-style compact groups; see "Group implementations in HDF5"). Accurately setting these parameters will help reduce wasted file space.

If a group is expected to have many links and to be stored in dense format, set `est_num_entries` to `0` (zero) for maximum efficiency. This will prevent the group from being created in the compact format.

See "Group implementations in HDF5" in the H5G API introduction for a discussion of the available types of HDF5 group structures.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
unsigned `est_num_entries`	IN: Estimated number of links to be inserted into group
unsigned `est_name_len`	IN: Estimated average length of link names

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE H5Pset_est_link_info_f(gcpl_id, est_num_entries, est_name_len, &
                                  hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id
                                    ! Group creation property list identifier
  INTEGER, INTENT(IN) :: est_num_entries
                                    ! Estimated number of links to be
                                    ! inserted into group
  INTEGER, INTENT(IN) :: est_name_len
                                    ! Estimated average length of link names
  INTEGER, INTENT(OUT) :: hdferr    ! Error code
                                    ! 0 on success and -1 on failure
END SUBROUTINE H5Pset_est_link_info_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_external

Signature:

herr_t `H5Pset_external`(*hid_t* `plist`, *const char* `*name`, *off_t* `offset`, *hsize_t* `size`)

Purpose:

Adds an external file to the list of external files.

Description:

The first call to `H5Pset_external` sets the *external storage* property in the property list, thus designating that the dataset will be stored in one or more non-HDF5 file(s) external to the HDF5 file. This call also adds the file `name` as the first file in the list of external files. Subsequent calls to the function add the named file as the next file in the list.

If a dataset is split across multiple files, then the files should be defined in order. The total size of the dataset is the sum of the `size` arguments for all the external files. If the total size is larger than the size of a dataset then the dataset can be extended (provided the data space also allows the extending).

The `size` argument specifies the number of bytes reserved for data in the external file. If `size` is set to `H5F_UNLIMITED`, the external file can be of unlimited size and no more files can be added to the external files list.

All of the external files for a given dataset must be specified with `H5Pset_external` *before* `H5Dcreate` is called to create the dataset. If one these files does not exist on the system when `H5Dwrite` is called to write data to it, the library will create the file.

Parameters:

hid_t `plist`	IN: Identifier of a dataset creation property list.
const char `*name`	IN: Name of an external file.
off_t `offset`	IN: Offset, in bytes, from the beginning of the file to the location in the file where the data starts.
hsize_t `size`	IN: Number of bytes reserved in the file for the data.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_external_f

```
SUBROUTINE h5pset_external_f(prp_id, name, offset,bytes, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of an external file
  INTEGER, INTENT(IN) :: offset         ! Offset, in bytes, from the
                                        ! beginning of the file to the
                                        ! location in the file where
                                        ! the data starts
  INTEGER(HSIZE_T), INTENT(IN) :: bytes ! Number of bytes reserved in
                                        ! the file for the data
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_external_f
```

Name: H5Pset_family_offset

Signature:

herr_t `H5Pset_family_offset` (*hid_t* `fapl_id`, *hsize_t* `offset`)

Purpose:

Sets offset property for low-level access to a file in a family of files.

Description:

`H5Pset_family_offset` sets the offset property in the file access property list `fapl_id` so that the user application can retrieve a file handle for low-level access to a particular member of a family of files. The file handle is retrieved with a separate call to `H5Fget_vfd_handle` (or, in special circumstances, to `H5FDget_vfd_handle`; see *Virtual File Layer* and *List of VFL Functions* in *HDF5 Technical Notes*).

The value of `offset` is an offset in bytes from the beginning of the HDF5 file, identifying a user-determined location within the HDF5 file. The file handle the user application is seeking is for the specific member-file in the associated family of files to which this offset is mapped.

Use of this function is only appropriate for an HDF5 file written as a family of files with the `FAMILY` file driver.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t `offset`	IN: Offset in bytes within the HDF5 file.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_family_offset_f

```
SUBROUTINE h5pset_family_offset_f(prp_id, offset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: prp_id   ! Property list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: offset   ! Offset in bytes
  INTEGER, INTENT(OUT)         :: hdferr   ! Error code
                                           ! 0 on success and -1 on failure

END SUBROUTINE h5pset_family_offset_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_fapl_core

Signature:

herr_t `H5Pset_fapl_core`(*hid_t* `fapl_id`, *size_t* `increment`, *hbool_t* `backing_store`)

Purpose:

Modifies the file access property list to use the `H5FD_CORE` driver.

Description:

`H5Pset_fapl_core` modifies the file access property list to use the `H5FD_CORE` driver.

The `H5FD_CORE` driver enables an application to work with a file in memory, speeding reads and writes as no disk access is made. File contents are stored only in memory until the file is closed. The `backing_store` parameter determines whether file contents are ever written to disk.

`increment` specifies the increment by which allocated memory is to be increased each time more memory is required.

While using `H5Fcreate` to create a core file, if the `backing_store` is set to `1` (`TRUE`), the file contents are flushed to a file with the same name as this core file when the file is closed or access to the file is terminated in memory.

The application is allowed to open an existing file with `H5FD_CORE` driver. While using `H5Fopen` to open an existing file, if the `backing_store` is set to `1` and the `flags` for `H5Fopen` is set to `H5F_ACC_RDWR`, any change to the file contents are saved to the file when the file is closed. If `backing_store` is set to `0` and the `flags` for `H5Fopen` is set to `H5F_ACC_RDWR`, any change to the file contents will be lost when the file is closed. If the `flags` for `H5Fopen` is set to `H5F_ACC_RDONLY`, no change to the file is allowed either in memory or on file.

Note:

Currently this driver cannot create or open family or multi files.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
size_t `increment`	IN: Size, in bytes, of memory increments.
hbool_t `backing_store`	IN: Boolean flag indicating whether to write the file contents to disk when the file is closed.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_fapl_core_f

```
SUBROUTINE h5pset_fapl_core_f(prp_id, increment, backing_store, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)  :: prp_id    ! Property list identifier
  INTEGER(SIZE_T), INTENT(IN) :: increment ! File block size in bytes
  LOGICAL, INTENT(IN) :: backing_store     ! Flag to indicate that entire
                                           ! file contents are flushed to
                                           ! a file with the same name as
                                           ! this core file
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fapl_core_f
```

History:

Release	**C**	**Fortran90**
1.6.0		The `backing_store` parameter has changed from *INTEGER* to *LOGICAL* to better match the C API.
1.4.0	Function introduced in this release.	

Name: H5Pset_fapl_direct

Signature:

herr_t `H5Pset_fapl_direct(` *hid_t* `fapl_id`, *size_t* `alignment`, *size_t* `block_size`, *size_t* `cbuf_size`)

Purpose:

Sets up use of the direct I/O driver.

Description:

`H5Pset_fapl_direct` sets the file access property list, `fapl_id`, to use the direct I/O driver, `H5FD_DIRECT`. With this driver, data is written to or read from the file synchronously without being cached by the system.

File systems usually require the data address in memory, the file address, and the size of the data to be aligned. The HDF5 Library's direct I/O driver is able to handle unaligned data, though that will consume some additional memory resources and may slow performance. To get better performance, use the system function `posix_memalign` to align the data buffer in memory and the HDF5 function `H5Pset_alignment` to align the data in the file. Be aware, however, that aligned data I/O may cause the HDF5 file to be bigger than the actual data size would otherwise require because the alignment may leave some holes in the file.

`alignment` specifies the required alignment boundary in memory.

`block_size` specifies the file system block size. A value of `0` (zero) means to use HDF5 Library's default value of 4KB.

`cbuf_size` specifies the copy buffer size.

Note:

On an SGI Altix Linux 2.6 system, the memory alignment must be a multiple of 512 bytes, and the file system block size is 4KB. The maximum size for the copy buffer has to be a multiple of the file system block size. The HDF5 Library's default maximum copy buffer size is 16MB. This copy buffer is used by the library's internal algorithm to copy data in fragments between an application's unaligned buffer and the file. The buffer's size may affect I/O performance.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
size_t `alignment`	IN: Required memory alignment boundary
size_t `block_size`	IN: File system block size
size_t `cbuf_size`	IN: Copy buffer size

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_fapl_direct_f

```
SUBROUTINE h5pset_fapl_direct_f(fapl_id, alignment, block_size, cbuf_size, &
                                hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: fapl_id ! File access property list identifier
  INTEGER(SIZE_T), INTENT(IN) :: alignment
                                        ! Required memory alignment boundary
  INTEGER(SIZE_T), INTENT(IN) :: block_size
                                        ! File system block size
  INTEGER(SIZE_T), INTENT(IN) :: cbuf_size
                                        ! Copy buffer size
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE H5Pset_fapl_direct_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_fapl_family

Signature:

herr_t `H5Pset_fapl_family` (*hid_t* `fapl_id`, *hsize_t* `memb_size`, *hid_t* `memb_fapl_id`)

Purpose:

Sets the file access property list to use the family driver.

Description:

`H5Pset_fapl_family` sets the file access property list identifier, `fapl_id`, to use the family driver.

`memb_size` is the size in bytes of each file member and is used only when creating a new file.

`memb_fapl_id` is the identifier of the file access property list to be used for each family member.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t `memb_size`	IN: Size in bytes of each file member.
hid_t `memb_fapl_id`	IN: Identifier of file access property list for each family member.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_fapl_family_f

```
SUBROUTINE h5pset_fapl_family_f(prp_id, imemb_size, memb_plist, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: prp_id    ! Property list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: memb_size ! Logical size, in bytes,
                                            ! of each family member
  INTEGER(HID_T), INTENT(IN) :: memb_plist  ! Identifier of the file
                                            ! access property list to be
                                            ! used for each family member
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fapl_family_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pset_fapl_log

Signature:

herr_t `H5Pset_fapl_log`(*hid_t* `fapl_id`, *const char* `*logfile`, *unsigned int* `flags`, *size_t* `buf_size`)

Purpose:

Sets up the use of the logging driver.

Description:

`H5Pset_fapl_log` modifies the file access property list to use the logging driver `H5FD_LOG`.

`logfile` is the name of the file in which the logging entries are to be recorded.

The actions to be logged are specified in the parameter `flags` using the pre-defined constants described in the following table. Multiple flags can be set through the use of an logical OR contained in parentheses. For example, logging read and write locations would be specified as `(H5FD_LOG_LOC_READ|H5FD_LOG_LOC_WRITE)`.

Flag	Description

`H5FD_LOG_LOC_READ`
`H5FD_LOG_LOC_WRITE`
`H5FD_LOG_LOC_SEEK`
	Track the location and length of every read, write, or seek operation.

`H5FD_LOG_LOC_IO`	Track all I/O locations and lengths. The logical equivalent of the following: `(H5FD_LOG_LOC_READ | H5FD_LOG_LOC_WRITE | H5FD_LOG_LOC_SEEK)`
`H5FD_LOG_FILE_READ` `H5FD_LOG_FILE_WRITE`	Track the number of times each byte is read or written.
`H5FD_LOG_FILE_IO`	Track the number of times each byte is read and written. The logical equivalent of the following: `(H5FD_LOG_FILE_READ | H5FD_LOG_FILE_WRITE)`
`H5FD_LOG_FLAVOR`	Track the type, or flavor, of information stored at each byte.

`H5FD_LOG_NUM_READ`
`H5FD_LOG_NUM_WRITE`
`H5FD_LOG_NUM_SEEK`
	Track the total number of read, write, or seek operations that occur.

`H5FD_LOG_NUM_IO`	Track the total number of all types of I/O operations. The logical equivalent of the following: `(H5FD_LOG_NUM_READ | H5FD_LOG_NUM_WRITE | H5FD_LOG_NUM_SEEK)`

`H5FD_LOG_TIME_OPEN`
`H5FD_LOG_TIME_READ`
`H5FD_LOG_TIME_WRITE`
`H5FD_LOG_TIME_SEEK`
`H5FD_LOG_TIME_CLOSE`
	Track the time spent in open, read, write, seek, or close operations.
	Not implemented in this release: open and read
	Partially implemented: write and seek
	Fully implemented: close

`H5FD_LOG_TIME_IO`	Track the time spent in each of the above operations. The logical equivalent of the following: `(H5FD_LOG_TIME_OPEN | H5FD_LOG_TIME_READ | H5FD_LOG_TIME_WRITE | H5FD_LOG_TIME_SEEK | H5FD_LOG_TIME_CLOSE)`
`H5FD_LOG_ALLOC`	Track the allocation of space in the file.
`H5FD_LOG_ALL`	Track everything. The logical equivalent of the following: `(H5FD_LOG_ALLOC | H5FD_LOG_TIME_IO | H5FD_LOG_NUM_IO | H5FD_LOG_FLAVOR |H5FD_LOG_FILE_IO | H5FD_LOG_LOC_IO)`

The logging driver can track the number of times each byte in the file is read from or written to (using `H5FD_LOG_FILE_READ` and `H5FD_LOG_FILE_WRITE`) and what kind of data is at that location (e.g., meta data, raw data; using `H5FD_LOG_FLAVOR`). This information is tracked in a buffer of size `buf_size`, which must be at least the size in bytes of the file to be logged.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
char `*logfile`	IN: Name of the log file.
unsigned int `flags`	IN: Flags specifying the types of logging activity.
size_t `buf_size`	IN: The size of the logging buffer.

Returns:

Returns non-negative if successful. Otherwise returns negative.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	The `verbosity` parameter has been removed. Two new parameters have been added: `flags` of type *unsigned* and `buf_size` of type *size_t*.
1.4.0	Function introduced in this release.

Last modified: 15 May 2009

Name: H5Pset_fapl_mpio

Signature:

herr_t `H5Pset_fapl_mpio`(*hid_t* `fapl_id`, *MPI_Comm* `comm`, *MPI_Info* `info`)

Purpose:

Stores MPI IO communicator information to the file access property list.

Description:

`H5Pset_fapl_mpio` stores the user-supplied MPI IO parameters `comm`, for communicator, and `info`, for information, in the file access property list `fapl_id`. That property list can then be used to create and/or open a file.

`H5Pset_fapl_mpio` is available only in the parallel HDF5 library and is not a collective function.

`comm` is the MPI communicator to be used for file open, as defined in `MPI_FILE_OPEN` of MPI-2. This function makes a duplicate of the communicator, so modifications to `comm` after this function call returns have no effect on the file access property list.

`info` is the MPI Info object to be used for file open, as defined in `MPI_FILE_OPEN` of MPI-2. This function makes a duplicate copy of the Info object, so modifications to the Info object after this function call returns will have no effect on the file access property list.

If the file access property list already contains previously-set communicator and Info values, those values will be replaced and the old communicator and Info object will be freed.

Note:

Raw dataset chunk caching is not currently supported when using this file driver in read/write mode. All calls to `H5Dread` and `H5Dwrite` will access the disk directly, and `H5Pset_cache` and `H5Pset_chunk_cache` will have no effect on performance.

Raw dataset chunk caching is supported when this driver is used in read-only mode.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
MPI_Comm `comm`	IN: MPI-2 communicator
MPI_Info `info`	IN: MPI-2 info object

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_fapl_mpio_f

```
SUBROUTINE h5pset_fapl_mpio_f(prp_id, comm, info, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(IN) :: comm           ! MPI communicator to be used for
                                        ! file open as defined in
                                        ! MPI_FILE_OPEN of MPI-2
  INTEGER, INTENT(IN) :: info           ! MPI info object to be used for
                                        ! file open as defined in
                                        ! MPI_FILE_OPEN of MPI-2
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fapl_mpio_f
```

History:

Release	**Change**
1.4.5	Handling of the MPI Communicator and Info object changed at this release. A duplicate of each of these is now stored in the property list instead of pointers to each.
1.4.0	C function introduced in this release.

Last modified: 15 April 2009

Name: H5Pset_fapl_mpiposix

Signature:

herr_t `H5Pset_fapl_mpiposix`(*hid_t* `fapl_id`, *MPI_Comm* `comm`, *hbool_t* `use_gpfs_hints`)

Purpose:

Stores MPI IO communicator information to a file access property list.

Description:

`H5Pset_fapl_mpiposix` stores the user-supplied MPI IO parameter `comm`, for communicator, in the file access property list `fapl_id`. That property list can then be used to create and/or open the file.

`H5Pset_fapl_mpiposix` is available only in the parallel HDF5 library and is not a collective function.

`comm` is the MPI communicator to be used for file open, as defined in `MPI_FILE_OPEN` of MPI-2. This function does not create a duplicated communicator. Modifications to `comm` after this function call returns may have an undetermined effect on the file access property list. Users should not modify the communicator while it is defined in a property list.

`use_gpfs_hints` specifies whether to attempt to use GPFS hints when accessing this file. A value of `TRUE` (or `1`) indicates that the hints should be used, if possible. A value of `FALSE` (or `0`) indicates that the hints should not be used.

Available GPFS hints are known to the HFD5 Library and are not user configurable. They may be used *only* with GPFS file systems and may improve file access for some applications; the user of a GPFS system is encouraged to experiment by running an application with and without this parameter set.

Note:

Raw dataset chunk caching is not currently supported when using this file driver in read/write mode. All calls to `H5Dread` and `H5Dwrite` will access the disk directly, and `H5Pset_cache` and `H5Pset_chunk_cache` will have no effect on performance.

Raw dataset chunk caching is supported when this driver is used in read-only mode.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
MPI_Comm `comm`	IN: MPI-2 communicator.
hbool_t `use_gpfs_hints`	IN: Use of GPFS hints.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_fapl_mpiposix_f

```
SUBROUTINE h5pset_fapl_mpiposix_f(prp_id, comm, use_gpfs, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(IN) :: comm           ! MPI communicator to be used
                                        ! for file open as defined in
                                        ! MPI_FILE_OPEN of MPI-2
  LOGICAL, INTENT(IN) :: use_gpfs
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5pset_fapl_mpiposix_f
```

History:

Release	**Change**
1.6.1	Fortran subroutine introduced in this release.
1.6.0	`use_gpfs_hints` parameter added.
1.6.0	C function introduced in this release.

Name: H5Pset_fapl_multi

Signature:

herr_t `H5Pset_fapl_multi`(*hid_t* `fapl_id`, *const H5FD_mem_t* *`memb_map`, *const hid_t* *`memb_fapl`, *const char * const* *`memb_name`, *const haddr_t* *`memb_addr`, *hbool_t* `relax`)

Purpose:

Sets up use of the multi-file driver.

Description:

`H5Pset_fapl_multi` sets the file access property list `fapl_id` to use the multi-file driver.

The multi-file driver enables different types of HDF5 data and metadata to be written to separate files. These files are viewed by the HDF5 library and the application as a single virtual HDF5 file with a single HDF5 file address space. The types of data that can be broken out into separate files include raw data, the superblock, B-tree data, global heap data, local heap data, and object headers. At the programmer's discretion, two or more types of data can be written to the same file while other types of data are written to separate files.

The array `memb_map` maps memory usage types to other memory usage types and is the mechanism that allows the caller to specify how many files are created. The array contains `H5FD_MEM_NTYPES` entries, which are either the value `H5FD_MEM_DEFAULT` or a memory usage type. The number of unique values determines the number of files that are opened.

The array `memb_fapl` contains a property list for each memory usage type that will be associated with a file.

The array `memb_name` should be a name generator (a printf-style format with a %s which will be replaced with the name passed to `H5FDopen`, usually from `H5Fcreate` or `H5Fopen`).

The array `memb_addr` specifies the offsets within the virtual address space, from `0` (zero) to `HADDR_MAX`, at which each type of data storage begins.

If `relax` is set to `TRUE` (or `1`), then opening an existing file for read-only access will not fail if some file members are missing. This allows a file to be accessed in a limited sense if just the meta data is available.

Default values for each of the optional arguments are as follows:

`memb_map`
: The default member map contains the value `H5FD_MEM_DEFAULT` for each element.

`memb_fapl`
: The default value is `H5P_DEFAULT` for each element.

`memb_name`
: The default string is `%s-`*`X`*`.h5` where *`X`* is one of the following letters:

`s`	for `H5FD_MEM_SUPER`
`b`	for `H5FD_MEM_BTREE`
`r`	for `H5FD_MEM_DRAW`
`g`	for `H5FD_MEM_GHEAP`
`l`	for `H5FD_MEM_LHEAP`
`o`	for `H5FD_MEM_OHDR`

`memb_addr`
The default value is `HADDR_UNDEF` for each element.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
const H5FD_mem_t *`memb_map`	IN: Maps memory usage types to other memory usage types.
const hid_t *`memb_fapl`	IN: Property list for each memory usage type.
*const char * const* *`memb_name`	IN: Name generator for names of member files.
const haddr_t *`memb_addr`	IN: The offsets within the virtual address space, from `0` (zero) to `HADDR_MAX`, at which each type of data storage begins.
hbool_t `relax`	IN: Allows read-only access to incomplete file sets when `TRUE`.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Example:

The following code sample sets up a multi-file access property list that partitions data into meta and raw files, each being one-half of the address:

```
H5FD_mem_t mt, memb_map[H5FD_MEM_NTYPES];
hid_t memb_fapl[H5FD_MEM_NTYPES];
const char *memb[H5FD_MEM_NTYPES];
haddr_t memb_addr[H5FD_MEM_NTYPES];

// The mapping...
for (mt=0; mt<H5FD_MEM_NTYPES; mt++) {
   memb_map[mt] = H5FD_MEM_SUPER;
}
memb_map[H5FD_MEM_DRAW] = H5FD_MEM_DRAW;

// Member information
memb_fapl[H5FD_MEM_SUPER] = H5P_DEFAULT;
memb_name[H5FD_MEM_SUPER] = "%s.meta";
memb_addr[H5FD_MEM_SUPER] = 0;

memb_fapl[H5FD_MEM_DRAW] = H5P_DEFAULT;
memb_name[H5FD_MEM_DRAW] = "%s.raw";
memb_addr[H5FD_MEM_DRAW] = HADDR_MAX/2;

hid_t fapl = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_multi(fapl, memb_map, memb_fapl,
                memb_name, memb_addr, TRUE);
```

Fortran90 Interface: h5pset_fapl_multi_f

```
SUBROUTINE h5pset_fapl_multi_f(prp_id, memb_map, memb_fapl, memb_name,
                               memb_addr, relax, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T),INTENT(IN)  :: prp_id     ! Property list identifier

  INTEGER,DIMENSION(0:H5FD_MEM_NTYPES_F-1),INTENT(IN)          :: memb_map
  INTEGER(HID_T),DIMENSION(0:H5FD_MEM_NTYPES_F-1),INTENT(IN)   :: memb_fapl
  CHARACTER(LEN=*),DIMENSION(0:H5FD_MEM_NTYPES_F-1),INTENT(IN) :: memb_name
  REAL, DIMENSION(0:H5FD_MEM_NTYPES_F-1), INTENT(IN)          :: memb_addr
            ! Numbers in the interval [0,1) (e.g. 0.0 0.1 0.5 0.2 0.3 0.4)
            ! real address in the file will be calculated as X*HADDR_MAX
```

```
        LOGICAL, INTENT(IN)  :: relax
        INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                                 ! 0 on success and -1 on failure
      END SUBROUTINE h5pset_fapl_multi_f
```

History:

Release	**C**
1.6.3	`memb_name` parameter type changed to *const char* const**.
1.4.0	Function introduced in this release.

Name: H5Pset_fapl_sec2

Signature:

herr_t `H5Pset_fapl_sec2`(*hid_t* `fapl_id`)

Purpose:

Sets the sec2 driver.

Description:

`H5Pset_fapl_sec2` modifies the file access property list to use the `H5FD_SEC2` driver.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_fapl_sec2_f

```
SUBROUTINE h5pset_fapl_sec2_f(prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)    :: prp_id  ! Property list identifier
  INTEGER, INTENT(OUT)          :: hdferr  ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fapl_sec2_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pset_fapl_split

Signature:

herr_t `H5Pset_fapl_split`(*hid_t* `fapl_id`, *const char* *`meta_ext`, *hid_t* `meta_plist_id`, *const char* *`raw_ext`, *hid_t* `raw_plist_id`)

Purpose:

Emulates the old split file driver.

Description:

`H5Pset_fapl_split` is a compatibility function that enables the multi-file driver to emulate the split driver from HDF5 Releases 1.0 and 1.2. The split file driver stored metadata and raw data in separate files but provided no mechanism for separating types of metadata.

`fapl_id` is a file access property list identifier.

`meta_ext` is the filename extension for the metadata file. The extension is appended to the name passed to `H5FDopen`, usually from `H5Fcreate` or `H5Fopen`, to form the name of the metadata file. If the string %s is used in the extension, it works like the name generator as in `H5Pset_fapl_multi`.

`meta_plist_id` is the file access property list identifier for the metadata file.

`raw_ext` is the filename extension for the raw data file. The extension is appended to the name passed to `H5FDopen`, usually from `H5Fcreate` or `H5Fopen`, to form the name of the rawdata file. If the string %s is used in the extension, it works like the name generator as in `H5Pset_fapl_multi`.

`raw_plist_id` is the file access property list identifier for the raw data file.

If a user wishes to check to see whether this driver is in use, the user must call `H5Pget_driver` and compare the returned value to the string `H5FD_MULTI`. A positive match will confirm that the multi driver is in use; HDF5 provides no mechanism to determine whether it was called as the special case invoked by `H5Pset_fapl_split`.

Parameters:

hid_t `fapl_id`,	IN: File access property list identifier.
const char *`meta_ext`,	IN: Metadata filename extension.
hid_t `meta_plist_id`,	IN: File access property list identifier for the metadata file.
const char *`raw_ext`,	IN: Raw data filename extension.
hid_t `raw_plist_id`	IN: File access property list identifier for the raw data file.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Example:

```
/* Example 1: Both metadata and rawdata files are in the same  */
/*    directory.   Use Station1-m.h5 and Station1-r.h5 as      */
/*    the metadata and rawdata files.                          */
hid_t fapl, fid;
fapl = H5Pcreate(H5P_FILE_ACCESS);
H5Pset_fapl_split(fapl, "-m.h5", H5P_DEFAULT, "-r.h5", H5P_DEFAULT);
fid=H5Fcreate("Station1",H5F_ACC_TRUNC,H5P_DEFAULT,fapl);

/* Example 2: metadata and rawdata files are in different      */
/*    directories.  Use PointA-m.h5 and /pfs/PointA-r.h5 as    */
/*    the metadata and rawdata files.                          */
hid_t fapl, fid;
```

```
      fapl = H5Pcreate(H5P_FILE_ACCESS);
      H5Pset_fapl_split(fapl, "-m.h5", H5P_DEFAULT, "/pfs/%s-r.h5", H5P_DEFAULT);
      fid=H5Fcreate("PointA",H5F_ACC_TRUNC,H5P_DEFAULT,fapl);
```

Fortran90 Interface: h5pset_fapl_split_f

```
SUBROUTINE h5pset_fapl_split_f(prp_id, meta_ext, meta_plist, raw_ext, &
                               raw_plist, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T),INTENT(IN)   :: prp_id     ! Property list identifier
  CHARACTER(LEN=*),INTENT(IN) :: meta_ext   ! Name of the extension for
                                            ! the metafile filename
  INTEGER(HID_T),INTENT(IN)   :: meta_plist ! Identifier of the meta file
                                            ! access property list
  CHARACTER(LEN=*),INTENT(IN) :: raw_ext    ! Name extension for the raw
                                            ! file filename
  INTEGER(HID_T),INTENT(IN)   :: raw_plist  ! Identifier of the raw file
                                            ! access property list
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fapl_split_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pset_fapl_stdio

Signature:

herr_t `H5Pset_fapl_stdio`(*hid_t* `fapl_id`)

Purpose:

Sets the standard I/O driver.

Description:

`H5Pset_fapl_stdio` modifies the file access property list to use the standard I/O driver, `H5FD_STDIO`.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_fapl_stdio_f

```
SUBROUTINE h5pset_fapl_stdio_f(prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)    :: prp_id  ! Property list identifier
  INTEGER, INTENT(OUT)          :: hdferr  ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fapl_stdio_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Pset_fapl_windows

Signature:

herr_t `H5Pset_fapl_windows`(*hid_t* `fapl_id`)

Purpose:

Sets the Windows I/O driver.

Description:

`H5Pset_fapl_windows` sets the default HDF5 Windows I/O driver on Windows systems.

Since the HDF5 Library uses this driver, `H5FD_WINDOWS`, by default on Windows systems, it is not normally necessary for a user application to call `H5Pset_fapl_windows`. While it is not recommended, there may be times when a user chooses to set a different HDF5 driver, such as the standard I/O driver (`H5FD_STDIO`) or the sec2 driver (`H5FD_SEC2`), in a Windows application. `H5Pset_fapl_windows` is provided so that the application can return to the Windows I/O driver when the time comes.

Only the Windows driver is tested on Windows systems; other drivers are used at the application's and the user's risk.

Furthermore, the Windows driver is tested and available only on Windows systems; it is not available on non-Windows systems.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_fclose_degree

Signature:

herr_t `H5Pset_fclose_degree`(*hid_t* `fapl_id`, *H5F_close_degree_t* `fc_degree`)

Purpose:

Sets the file close degree.

Description:

`H5Pset_fclose_degree` sets the file close degree property `fc_degree` in the file access property list `fapl_id`.

The value of `fc_degree` determines how aggressively `H5Fclose` deals with objects within a file that remain open when `H5Fclose` is called to close that file. `fc_degree` can have any one of four valid values:

Degree name	`H5Fclose` behavior with no open object in file	`H5Fclose` behavior with open object(s) in file
`H5F_CLOSE_WEAK`	Actual file is closed.	Access to file identifier is terminated; actual file close is delayed until all objects in file are closed
`H5F_CLOSE_SEMI`	Actual file is closed.	Function returns FAILURE
`H5F_CLOSE_STRONG`	Actual file is closed.	All open objects remaining in the file are closed then file is closed
`H5F_CLOSE_DEFAULT`	The VFL driver chooses the behavior. Currently, all VFL drivers set this value to `H5F_CLOSE_WEAK`, except for the MPI-I/O driver, which sets it to `H5F_CLOSE_SEMI`.	

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
H5F_close_degree_t `fc_degree`	IN: Pointer to a location containing the file close degree property, the value of `fc_degree`.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_fclose_degree_f

```
SUBROUTINE h5pset_fclose_degree_f(fapl_id, degree, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: fapl_id  ! File access property list identifier
  INTEGER, INTENT(IN) :: degree          ! Info about file close behavior
                                         ! Possible values:
                                         !    H5F_CLOSE_DEFAULT_F
                                         !    H5F_CLOSE_WEAK_F
                                         !    H5F_CLOSE_SEMI_F
                                         !    H5F_CLOSE_STRONG_F
```

```
    INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                            ! 0 on success and -1 on failure
  END SUBROUTINE h5pset_fclose_degree_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_fill_time

Signature:

herr_t `H5Pset_fill_time`(*hid_t* `plist_id`, *H5D_fill_time_t* `fill_time`)

Purpose:

Sets the time when fill values are written to a dataset.

Description:

`H5Pset_fill_time` sets up the timing for writing fill values to a dataset. This property is set in the dataset creation property list `plist_id`.

Timing is specified in `fill_time` with one of the following values:

`H5D_FILL_TIME_IFSET`	Write fill values to the dataset when storage space is allocated only if there is a user-defined fill value, i.e., one set with H5Pset_fill_value. (Default)
`H5D_FILL_TIME_ALLOC`	Write fill values to the dataset when storage space is allocated.
`H5D_FILL_TIME_NEVER`	Never write fill values to the dataset.

Note:

`H5Pset_fill_time` is designed for coordination with the dataset fill value and dataset storage allocation time properties, set with the functions `H5Pset_fill_value` and `H5Pset_alloc_time`.

See H5Dcreate for further cross-references.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
H5D_fill_time_t `fill_time`	IN: When to write fill values to a dataset.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_fill_time_f

```
SUBROUTINE h5pset_fill_time_f(plist_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! Dataset creation property
                                         ! list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: flag   ! File time flag
                                         ! Possible values are:
                                         !    H5D_FILL_TIME_ERROR_F
                                         !    H5D_FILL_TIME_ALLOC_F
                                         !    H5D_FILL_TIME_NEVER_F
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fill_time_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_fill_value

Signature:

herr_t `H5Pset_fill_value`(*hid_t* `plist_id`, *hid_t* `type_id`, *const void* *`value`)

Purpose:

Sets the fill value for a dataset.

Description:

`H5Pset_fill_value` sets the fill value for a dataset in the dataset creation property list.

`value` is interpreted as being of datatype `type_id`. This datatype may differ from that of the dataset, but the HDF5 library must be able to convert `value` to the dataset datatype when the dataset is created.

The default fill value is `0` (zero), which is interpreted according to the actual dataset datatype.

Setting `value` to `NULL` indicates that the fill value is to be undefined.

Notes:

Applications sometimes write data only to portions of an allocated dataset. It is often useful in such cases to fill the unused space with a known fill value. This function allows the user application to set that fill value; the functions H5Dfill and H5Pset_fill_time, respectively, provide the ability to apply the fill value on demand or to set up its automatic application.

A fill value should be defined so that it is appropriate for the application. While the HDF5 default fill value is `0` (zero), it is often appropriate to use another value. It might be useful, for example, to use a value that is known to be impossible for the application to legitimately generate.

`H5Pset_fill_value` is designed to work in concert with `H5Pset_alloc_time` and `H5Pset_fill_time`. `H5Pset_alloc_time` and `H5Pset_fill_time` govern the timing of dataset storage allocation and fill value write operations and can be important in tuning application performance.

See H5Dcreate for further cross-references.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
hid_t `type_id`,	IN: Datatype of `value`.
const void *`value`	IN: Pointer to buffer containing value to use as fill value.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_fill_value_f

```
SUBROUTINE h5pset_fill_value_f(prp_id, type_id, fillvalue, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier of fill
                                        ! value datatype (in memory)
  TYPE(VOID), INTENT(IN) :: fillvalue   ! Fillvalue
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fill_value_f
```

Name: H5Pset_filter

Signature:

herr_t `H5Pset_filter`(*hid_t* `plist`, *H5Z_filter_t* `filter_id`, *unsigned int* `flags`, *size_t* `cd_nelmts`, *const unsigned int* `cd_values[]`)

Purpose:

Adds a filter to the filter pipeline.

Description:

`H5Pset_filter` adds the specified `filter_id` and corresponding properties to the end of an output filter pipeline. If `plist` is a dataset creation property list, the filter is added to the permanent filter pipeline; if `plist` is a dataset transfer property list, the filter is added to the transient filter pipeline.

The array `cd_values` contains `cd_nelmts` integers which are auxiliary data for the filter. The integer values will be stored in the dataset object header as part of the filter information.

The `flags` argument is a bit vector with the following fields specifying certain general properties of the filter:

`H5Z_FLAG_OPTIONAL`	If this bit is set then the filter is optional. If the filter fails (see below) during an `H5Dwrite` operation then the filter is just excluded from the pipeline for the chunk for which it failed; the filter will not participate in the pipeline during an `H5Dread` of the chunk. This is commonly used for compression filters: if the filter result would be larger than the input, then the compression filter returns failure and the uncompressed data is stored in the file. If this bit is clear and a filter fails, then `H5Dwrite` or `H5Dread` also fails. This flag should not be set for the Fletcher32 checksum filter as it will bypass the checksum filter without reporting checksum errors to an application.

The `filter_id` parameter specifies the filter to be set. Valid filter identifiers are as follows:

`H5Z_FILTER_DEFLATE`	Data compression filter, employing the gzip algorithm
`H5Z_FILTER_SHUFFLE`	Data shuffling filter
`H5Z_FILTER_FLETCHER32`	Error detection filter, employing the Fletcher32 checksum algorithm
`H5Z_FILTER_SZIP`	Data compression filter, employing the SZIP algorithm

Also see H5Pset_edc_check and H5Pset_filter_callback.

Notes:

This function currently supports only the permanent filter pipeline; `plist` must be a dataset creation property list.

If multiple filters are set for a property list, they will be applied to each chunk in the order in which they were set.

See Also:

For a discussion of optional versus required filter behavior, see “Filter Behavior in HDF5.”

Parameters:

hid_t `plist`	IN: Property list identifier.
H5Z_filter_t `filter_id`	IN: Filter identifier for the filter to be added to the pipeline.
unsigned int `flags`	IN: Bit vector specifying certain general properties of the filter.
size_t `cd_nelmts`	IN: Number of elements in `cd_values`.
const unsigned int `cd_values[]`	IN: Auxiliary data for the filter.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_filter_f

```
SUBROUTINE h5pset_filter_f(prp_id, filter, flags, cd_nelmts, cd_values,  hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER, INTENT(IN) :: filter         ! Filter to be added to the pipeline
  INTEGER, INTENT(IN) :: flags          ! Bit vector specifying certain
                                        ! general properties of the filter
  INTEGER(SIZE_T), INTENT(IN) :: cd_nelmts
                                        ! Number of elements in cd_values
  INTEGER, DIMENSION(*), INTENT(IN) :: cd_values
                                        ! Auxiliary data for the filter
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_filter_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_filter_callback

Signature:

herr_t `H5Pset_filter_callback`(*hid_t* `plist`, *H5Z_filter_func_t* `func`, *void* *`op_data`)

Purpose:

Sets user-defined filter callback function.

Description:

`H5Pset_filter_callback` sets the user-defined filter callback function `func` in the dataset transfer property list `plist`.

The parameter `op_data` is a pointer to user-defined input data for the callback function and will be passed through to the callback function.

The callback function `func` defines the actions an application is to take when a filter fails. The function prototype is as follows:

`typedef` *H5Z_cb_return_t* (`H5Z_filter_func_t`) (*H5Z_filter_t* `filter_id`, *void* *`buf`, *size_t* `buf_size`, *void* *`op_data`)

where `filter_id` indicates which filter has failed, `buf` and `buf_size` are used to pass in the failed data, and `op_data` is the required input data for this callback function.

Valid callback function return values are `H5Z_CB_FAIL` and `H5Z_CB_CONT`.

Parameters:

hid_t `plist`	IN: Dataset transfer property list identifier.
H5Z_filter_func_t `func`	IN: User-defined filter callback function.
void *`op_data`	IN: User-defined input data for the callback function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_fletcher32

Signature:

herr_t `H5Pset_fletcher32`(*hid_t* `plist`)

Purpose:

Sets up use of the Fletcher32 checksum filter.

Description:

`H5Pset_fletcher32` sets the Fletcher32 checksum filter in the dataset creation property list `plist`.

Note:

The initial error detection implementation supports error detection for chunked datasets only.

Parameters:

hid_t `plist`	IN: Dataset creation property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_fletcher32_f

```
SUBROUTINE h5pset_fletcher32_f(prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Dataset creation property list
                                        ! identifier
  INTEGER, INTENT(OUT)       :: hdferr  ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_fletcher32_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_gc_references

Signature:

herr_t `H5Pset_gc_reference`(*hid_t* `plist`, *unsigned* `gc_ref`)

Purpose:

Sets garbage collecting references flag.

Description:

`H5Pset_gc_references` sets the flag for garbage collecting references for the file.

Dataset region references and other reference types use space in an HDF5 file's global heap. If garbage collection is on and the user passes in an uninitialized value in a reference structure, the heap might get corrupted. When garbage collection is off, however, and the user re-uses a reference, the previous heap block will be orphaned and not returned to the free heap space.

When garbage collection is on, the user must initialize the reference structures to 0 or risk heap corruption.

The default value for garbage collecting references is off.

Parameters:

hid_t `plist`	IN: File access property list identifier.
unsigned `gc_ref`	IN: Flag setting reference garbage collection to on (`1`) or off (`0`).

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_gc_references_f

```
SUBROUTINE h5pset_gc_references_f (prp_id, gc_reference, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(IN) :: gc_reference  ! Flag for garbage collecting
                                       ! references for the file
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pset_gc_references_f
```

Name: H5Pset_hyper_vector_size

Signature:

herr_t `H5Pset_hyper_vector_size`(*hid_t* `dxpl_id`, *size_t* `vector_size`)

Purpose:

Sets number of I/O vectors to be read/written in hyperslab I/O.

Description:

`H5Pset_hyper_vector_size` sets the number of I/O vectors to be accumulated in memory before being issued to the lower levels of the HDF5 library for reading or writing the actual data.

The *I/O vectors* are hyperslab offset and length pairs and are generated during hyperslab I/O.

The number of I/O vectors is passed in `vector_size` to be set in the dataset transfer property list `dxpl_id`. `vector_size` must be greater than `1` (one).

`H5Pset_hyper_vector_size` is an I/O optimization function; increasing `vector_size` should provide better performance, but the library will use more memory during hyperslab I/O. The default value of `vector_size` is `1024`.

Parameters:

hid_t `dxpl_id`	IN: Dataset transfer property list identifier.
size_t `vector_size`	IN: Number of I/O vectors to accumulate in memory for I/O operations. Must be greater than `1` (one). Default value: `1024`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_hyper_vector_size_f

```
SUBROUTINE h5pset_hyper_vector_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! Dataset transfer property list
                                         ! identifier
  INTEGER(SIZE_T), INTENT(IN) :: size    ! Vector size
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_hyper_vector_size_f
```

History:

Release	C
1.6.0	Function introduced in this release.

Name: H5Pset_istore_k

Signature:

herr_t `H5Pset_istore_k`(*hid_t* `plist`, *unsigned* `ik`)

Purpose:

Sets the size of the parameter used to control the B-trees for indexing chunked datasets.

Description:

`H5Pset_istore_k` sets the size of the parameter used to control the B-trees for indexing chunked datasets. This function is only valid for file creation property lists.

`ik` is one half the rank of a tree that stores chunked raw data. On average, such a tree will be 75% full, or have an average rank of 1.5 times the value of `ik`.

Parameters:

hid_t `plist`	IN: Identifier of property list to query.
unsigned `ik`	IN: 1/2 rank of chunked storage B-tree.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_istore_k_f

```
SUBROUTINE h5pset_istore_k_f (prp_id, ik, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(IN) :: ik            ! 1/2 rank of chunked storage B-tree
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pset_istore_k_f
```

History:

Release	**C**
1.6.4	`ik` parameter type changed to *unsigned*.

Name: H5Pset_layout

Signature:

herr_t `H5Pset_layout`(*hid_t* `plist`, *H5D_layout_t* `layout`)

Purpose:

Sets the type of storage used to store the raw data for a dataset.

Description:

`H5Pset_layout` sets the type of storage used to store the raw data for a dataset. This function is only valid for dataset creation property lists.

Valid values for `layout` are:

H5D_COMPACT
: Store raw data in the dataset object header in file. This should only be used for very small amounts of raw data. The current limit is approximately 64K (HDF5 Release 1.6).

H5D_CONTIGUOUS
: Store raw data separately from the object header in one large chunk in the file.

H5D_CHUNKED
: Store raw data separately from the object header as chunks of data in separate locations in the file.

Note that a compact storage layout may affect writing data to the dataset with parallel applications. See note in H5Dwrite documentation for details.

Parameters:

hid_t `plist`	IN: Identifier of property list to query.
H5D_layout_t `layout`	IN: Type of storage layout for raw data.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_layout_f

```
SUBROUTINE h5pset_layout_f (prp_id, layout, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(IN) :: layout        ! Type of storage layout for raw data
                                       ! Possible values are:
                                       !    H5D_COMPACT_F
                                       !    H5D_CONTIGUOUS_F
                                       !    H5D_CHUNKED_F
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pset_layout_f
```

Name: H5Pset_libver_bounds

Signature:

herr_t `H5Pset_libver_bounds(` *hid_t* `fapl_id`, *H5F_libver_t* `low`, *H5F_libver_t* `high`)

Purpose:

Sets bounds on library versions, and indirectly format versions, to be used when creating objects.

Description:

`H5Pset_libver_bounds` controls the versions of the object formats that will be used when creating objects in a file. The object format versions are determined indirectly from the HDF5 Library versions specified in the call.

This property is set in the file access property list specified by `fapl_id`.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier
H5F_libver_t `low`	IN: The earliest version of the library that will be used for writing objects. The library version indirectly specifies the earliest object format version that can be used when creating objects in the file. Currently, `low` must be one of two pre-defined values: Setting `low` to `H5F_LIBVER_EARLIEST` will result in objects being created using the *earliest possible* format for each object. Note that *earliest possible* is different from *earliest*, as some features introduced in library versions later than 1.0.0 resulted in updates to object formats. With `low=H5F_LIBVER_EARLIEST`, if the application creates an object that requires a feature introduced in library versions later than 1.0.0, the earliest possible version that supports the requested feature will be used. The upper bound on the range of possible library versions used to create the object is controlled by the `high` parameter. This is the default behavior of the library if `H5Pset_libver_bounds` is not called. Setting `low` to `H5F_LIBVER_LATEST` will result in objects being created using the *latest available* format for each object. This setting means that objects will be created with the latest format versions available (within the range of library versions specified in the call), and can take advantage of the latest features and performance enhancements. Objects written with the `H5F_LIBVER_LATEST` setting for `low` may be accessible to a smaller range of library versions than would be the case if the `H5F_LIBVER_EARLIEST` value had been used. The upper bound on the range of possible library versions used to create the object is controlled by the `high` parameter.
H5F_libver_t `high`	IN: The latest version of the library that will be used for writing objects. The library version indirectly specifies the latest object format version that can be used when creating objects in the file.

Currently, `high` must be set to the pre-defined value `H5F_LIBVER_LATEST`. `H5F_LIBVER_LATEST` corresponds to the version of the HDF5 Library in use.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_libver_bounds_f

```
SUBROUTINE h5pset_libver_bounds_f(fapl_id, low, high, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: fapl_id ! File access property list identifier
  INTEGER, INTENT(IN) :: low ! The earliest version of the library that
                             ! will be used for writing objects.
                             ! Currently, low must be either:
                             !        H5F_LIBVER_EARLIEST_F
                             !        H5F_LIBVER_LATEST_F
    INTEGER, INTENT(IN) :: high
                             ! The latest version of the library that will be
                             ! used for writing objects.
                             ! Currently, high must set to:
                             !            H5F_LIBVER_LATEST_F
    INTEGER, INTENT(OUT) :: hdferr
                             ! Error code
                             ! 0 on success and -1 on failure
END SUBROUTINE h5pset_libver_bounds_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_link_creation_order

Signature:

herr_t `H5Pset_link_creation_order`(*hid_t* `gcpl_id`, *unsigned* `crt_order_flags`)

Purpose:

Sets creation order tracking and indexing for links in a group.

Description:

`H5Pset_link_creation_order` sets flags in a group creation property list, `gcpl_id`, for tracking and/or indexing links on creation order.

The following flags are passed in `crt_order_flags`:

`H5P_CRT_ORDER_TRACKED`	Specifies to track creation order.
`H5P_CRT_ORDER_INDEXED`	Specifies to index links in the group on creation order.

If only `H5P_CRT_ORDER_TRACKED` is set, HDF5 will track link creation order in any group created with the group creation property list `gcpl_id`. If both `H5P_CRT_ORDER_TRACKED` and `H5P_CRT_ORDER_INDEXED` are set, HDF5 will track link creation order in the group and index links on that property.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
unsigned `crt_order_flags`	IN: Creation order flag(s)

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_link_creation_order_f

```
SUBROUTINE h5pset_link_creation_order_f(gcpl_id, crt_order_flags, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id  ! File access property list identifier
  INTEGER, INTENT(IN) :: crt_order_flags ! Creation order flag(s)
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_link_creation_order_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_link_phase_change

Signature:

herr_t `H5Pset_link_phase_change`(*hid_t* `gcpl_id`, *unsigned* `max_compact`, *unsigned* `min_dense`)

Purpose:

Sets the parameters for conversion between compact and dense groups.

Description:

`H5Pset_link_phase_change` sets the maximum number of entries for a *compact* group and the minimum number of links to allow before converting a *dense* group to back to the compact format.

`max_compact` is the maximum number of links to store as header messages in the group header as before converting the group to the dense format. Groups that are in compact format and in which the exceed this number of links rises above this threshold are automatically converted to dense format.

`min_dense` is the minimum number of links to store in the dense format. Groups which are in dense format and in which the number of links falls below this theshold are automatically converted to compact format.

See “Group implementations in HDF5” in the H5G API introduction for a discussion of the available types of HDF5 group structures.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
unsigned `max_compact`	IN: Maximum number of links for compact storage *(Default: 8)*
unsigned `min_dense`	IN: Minimum number of links for dense storage *(Default: 6)*

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_link_phase_change_f

```
SUBROUTINE h5pset_link_phase_change_f(gcpl_id, max_compact, min_dense, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id ! Group creation property list id
  INTEGER, INTENT(IN) :: max_compact    ! Maximum number of attributes to be
                                        ! stored in compact storage
  INTEGER, INTENT(IN) :: min_dense      ! Minimum number of attributes to be
                                        ! stored in dense storage
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_link_phase_change_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_local_heap_size_hint

Signature:

herr_t `H5Pset_local_heap_size_hint(` *hid_t* `gcpl_id`, *size_t* `size_hint`)

Purpose:

Specifies the anticipated maximum size of a local heap.

Description:

`H5Pset_local_heap_size_hint` is used with original-style HDF5 groups (see "Motivation" below) to specify the anticipated maximum local heap size, `size_hint`, for groups created with the group creation property list `gcpl_id`. The HDF5 Library then uses `size_hint` to allocate contiguous local heap space in the file for each group created with `gcpl_id`.

For groups with many members or very few members, an appropriate initial value of `size_hint` would be the anticipated number of group members times the average length of group member names, plus a small margin:

```
size_hint = max_number_of_group_members  *
            (average_length_of_group_member_link_names + 2)
```

If it is known that there will be groups with zero members, the use of a group creation property list with `size_hint` set to to `1` (one) will guarantee the smallest possible local heap for each of those groups.

Setting `size_hint` to zero (`0`) causes the library to make a reasonable estimate for the default local heap size.

Motivation:

In situations where backward-compatibility is required, specifically, when libraries prior to HDF5 Release 1.8.0 may be used to read the file, groups must be created and maintained in the original style. This is HDF5's default behavior. If backward compatibility with pre-1.8.0 libraries is not a concern, greater efficiencies can be obtained with the new-format compact and indexed groups. See "Group implementations in HDF5" in the H5G API introduction.

`H5Pset_local_heap_size_hint` is useful for tuning file size when files contain original-style groups with either zero members or very large numbers of members.

The original style of HDF5 groups, the only style available prior to HDF5 Release 1.8.0, was well-suited for moderate-sized groups but was not optimized for either very small or very large groups. This original style remains the default, but two new group implementations were introduced in HDF5 Release 1.8.0: compact groups to accomodate zero to small numbers of members and indexed groups for thousands or tens of thousands of members ... or millions, if that's what your application requires.

The local heap size hint, `size_hint`, is a performance tuning parameter *for original-style groups*. As indicated above, an HDF5 group may have zero, a handful, or tens of thousands of members. Since the original style of HDF5 groups stores the metadata for all of these group members in a uniform format in a local heap, the size of that metadata (and hence, the size of the local heap) can vary wildly from group to group. To intelligently allocate space and to avoid unnecessary fragmentation of the local heap, it can be valuable to provide the library with a hint as to the local heap's likely eventual size. This can be particularly valuable when it is known that a group will eventually have a great many members. It can also be useful in conserving space in a file when it is known that certain groups will never have any members.

Parameters:

hid_t `gcpl_id`	IN: Group creation property list identifier
size_t `size_hint`	IN: Anticipated maximum size in bytes of local heap

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_local_heap_size_hint_f

```
SUBROUTINE h5pset_local_heap_size_hint_f(gcpl_id, size_hint, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: gcpl_id    ! Group creation property list id
  INTEGER(SIZE_T), INTENT(IN) :: size_hint ! Hint for size of local heap
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5pset_local_heap_size_hint_f
```

History:

Release	**Change**
1.8.0	Function introduced in this release.

Last modified: 18 May 2009

Name: H5Pset_mdc_config

Signature:

herr_t `H5Pset_mdc_config`(*hid_t* `plist_id`, *H5AC_cache_config_t* *`config_ptr`)

Purpose:

Set the initial metadata cache configuration in the indicated File Access Property List to the supplied value.

Description:

H5Pset_mdc_config attempts to set the initial metadata cache configuration to the supplied value. It will fail if an invalid configuration is detected. This configuration is used when the file is opened.

See the overview of the metadata cache in the special topics section of the user manual for details on what is being configured. If you haven't read and understood that documentation, you really shouldn't be using this API call.

Parameters:

hid_t `plist_id`	IN: Identifier of the file access property list.
H5AC_cache_config_t *`config_ptr`	IN: Pointer to the instance of H5AC_cache_config_t containing the desired configuration. The fields of this structure are discussed below:
General configuration section:	
int `version`	IN: Integer field indicating the the version of the H5AC_cache_config_t in use. This field should be set to H5AC__CURR_CACHE_CONFIG_VERSION (defined in H5ACpublic.h).
hbool_t `rpt_fcn_enabled`	IN: Boolean flag indicating whether the adaptive cache resize report function is enabled. This field should almost always be set to FALSE. Since resize algorithm activity is reported via stdout, it MUST be set to FALSE on Windows machines. The report function is not supported code, and can be expected to change between versions of the library. Use it at your own risk.
hbool_t `open_trace_file`	IN: Boolean field indicating whether the `trace_file_name` field should be used to open a trace file for the cache. The trace file is a debuging feature that allows the capture of top level metadata cache requests for purposes of debugging and/or optimization. This field should normally be set to `FALSE`, as trace file collection imposes considerable overhead.

	This field should only be set to `TRUE` when the `trace_file_name` contains the full path of the desired trace file, and either there is no open trace file on the cache, or the `close_trace_file` field is also `TRUE`. The trace file feature is unsupported unless used at the direction of THG. It is intended to allow THG to collect a trace of cache activity in cases of occult failures and/or poor performance seen in the field, so as to aid in reproduction in the lab. If you use it absent the direction of THG, you are on your own.
hbool_t `close_trace_file`	IN: Boolean field indicating whether the current trace file (if any) should be closed. See the above comments on the `open_trace_file` field. This field should be set to `FALSE` unless there is an open trace file on the cache that you wish to close. The trace file feature is unsupported unless used at the direction of THG. It is intended to allow THG to collect a trace of cache activity in cases of occult failures and/or poor performance seen in the field, so as to aid in reproduction in the lab. If you use it absent the direction of THG, you are on your own.
char `trace_file_name[]`	IN: Full path of the trace file to be opened if the `open_trace_file` field is `TRUE`. In the parallel case, an ascii representation of the MPI rank of the process will be appended to the file name to yield a unique trace file name for each process. The length of the path must not exceed `H5AC__MAX_TRACE_FILE_NAME_LEN` characters. The trace file feature is unsupported unless used at the direction of THG. It is intended to allow THG to collect a trace of cache activity in cases of occult failures and/or poor performance seen in the field, so as to aid in reproduction in the lab. If you use it absent the direction of THG, you are on your own.
hbool_t `evictions_enabled`	IN: A boolean flag indicating whether evictions from the metadata cache are enabled. This flag is initially set to `TRUE`.

	In rare circumstances, the raw data throughput requirements may be so high that the user wishes to postpone metadata writes so as to reserve I/O throughput for raw data. The `evictions_enabled` field exists to allow this. However, this is an extreme step, and you have no business doing it unless you have read the User Guide section on metadata caching, and have considered all other options carefully. The `evictions_enabled` field may not be set to `FALSE` unless all adaptive cache resizing code is disabled via the `incr_mode`, `flash_incr_mode`, and `decr_mode` fields. When this flag is set to `FALSE`, the metadata cache will not attempt to evict entries to make space for new entries, and thus will grow without bound. Evictions will be re-enabled when this field is set back to `TRUE`. This should be done as soon as possible.
hbool_t `set_initial_size`	IN: Boolean flag indicating whether the cache should be created with a user specified initial size.
size_t `initial_size`	IN: If set_initial_size is TRUE, initial_size must contains the desired initial size in bytes. This value must lie in the closed interval [min_size, max_size]. (see below)
double `min_clean_fraction`	IN: This field specifies the minimum fraction of the cache that must be kept either clean or empty. The value must lie in the interval [0.0, 1.0]. 0.01 is a good place to start in the serial case. In the parallel case, a larger value is needed -- see the overview of the metadata cache in the “HDF5 Special Topics” section of the *HDF5 User’s Guide* for details.
size_t `max_size`	IN: Upper bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
size_t `min_size`	IN: Lower bound (in bytes) on the range of values that the adaptive cache resize code can select as the maximum cache size.
long int `epoch_length`	IN: Number of cache accesses between runs of the adaptive cache resize code. 50,000 is a good starting number.

Increment configuration section:

enum H5C_cache_incr_mode `incr_mode`	IN: Enumerated value indicating the operational mode of the automatic cache size increase code. At present, only two values are legal: H5C_incr__off: Automatic cache size increase is disabled, and the remaining increment fields are ignored. H5C_incr__threshold: Automatic cache size increase is enabled using the hit rate threshold algorithm.
double `lower_hr_threshold`	IN: Hit rate threshold used by the hit rate threshold cache size increment algorithm. When the hit rate over an epoch is below this threshold and the cache is full, the maximum size of the cache is multiplied by increment (below), and then clipped as necessary to stay within max_size, and possibly max_increment. This field must lie in the interval [0.0, 1.0]. 0.8 or 0.9 is a good place to start.
double `increment`	IN: Factor by which the hit rate threshold cache size increment algorithm multiplies the current cache max size to obtain a tentative new cache size. The actual cache size increase will be clipped to satisfy the max_size specified in the general configuration, and possibly max_increment below. The parameter must be greater than or equal to 1.0 -- 2.0 is a reasonable value. If you set it to 1.0, you will effectively disable cache size increases.
hbool_t `apply_max_increment`	IN: Boolean flag indicating whether an upper limit should be applied to the size of cache size increases.
size_t `max_increment`	IN: Maximum number of bytes by which cache size can be increased in a single step -- if applicable.
enum H5C_cache_flash_incr_mode `flash_incr_mode`	IN: Enumerated value indicating the operational mode of the flash cache size increase code. At present, only the following values are legal:

	H5C_flash_incr__off: Flash cache size increase is disabled. H5C_flash_incr__add_space: Flash cache size increase is enabled using the add space algorithm.
double `flash_threshold`	IN: The factor by which the current maximum cache size is multiplied to obtain the minimum size entry / entry size increase which may trigger a flash cache size increase. At present, this value must lie in the range [0.1, 1.0].
double `flash_multiple`	IN: The factor by which the size of the triggering entry / entry size increase is multiplied to obtain the initial cache size increment. This increment may be reduced to reflect existing free space in the cache and the `max_size` field above. At present, this field must lie in the range [0.1, 10.0].
Decrement configuration section:	
enum H5C_cache_decr_mode `decr_mode`	IN: Enumerated value indicating the operational mode of the automatic cache size decrease code. At present, the following values are legal: H5C_decr__off: Automatic cache size decrease is disabled. H5C_decr__threshold: Automatic cache size decrease is enabled using the hit rate threshold algorithm. H5C_decr__age_out: Automatic cache size decrease is enabled using the ageout algorithm. H5C_decr__age_out_with_threshold: Automatic cache size decrease is enabled using the ageout with hit rate threshold algorithm

double `upper_hr_threshold`	IN: Hit rate threshold for the hit rate threshold and ageout with hit rate threshold cache size decrement algorithms. When decr_mode is H5C_decr__threshold, and the hit rate over a given epoch exceeds the supplied threshold, the current maximum cache size is multiplied by decrement to obtain a tentative new (and smaller) maximum cache size. When decr_mode is H5C_decr__age_out_with_threshold, there is no attempt to find and evict aged out entries unless the hit rate in the previous epoch exceeded the supplied threshold. This field must lie in the interval [0.0, 1.0]. For H5C_incr__threshold, .9995 or .99995 is a good place to start. For H5C_decr__age_out_with_threshold, .999 might be more useful.
double `decrement`	IN: In the hit rate threshold cache size decrease algorithm, this parameter contains the factor by which the current max cache size is multiplied to produce a tentative new cache size. The actual cache size decrease will be clipped to satisfy the min_size specified in the general configuration, and possibly max_decrement below. The parameter must be be in the interval [0.0, 1.0]. If you set it to 1.0, you will effectively disable cache size decreases. 0.9 is a reasonable starting point.
hbool_t `apply_max_decrement`	IN: Boolean flag indicating whether an upper limit should be applied to the size of cache size decreases.
size_t `max_decrement`	IN: Maximum number of bytes by which the maximum cache size can be decreased in any single step -- if applicable.

int `epochs_before_eviction`	IN: In the ageout based cache size reduction algorithms, this field contains the minimum number of epochs an entry must remain unaccessed in cache before the cache size reduction algorithm tries to evict it. 3 is a reasonable value.
hbool_t `apply_empty_reserve`	IN: Boolean flag indicating whether the ageout based decrement algorithms will maintain a empty reserve when decreasing cache size.
double `empty_reserve`	IN: Empty reserve as a fraction of maximum cache size if applicable. When so directed, the ageout based algorithms will not decrease the maximum cache size unless the empty reserve can be met. The parameter must lie in the interval [0.0, 1.0]. 0.1 or 0.05 is a good place to start.
Parallel configuration section:	
int `dirty_bytes_threshold`	IN: Threshold number of bytes of dirty metadata generation for triggering synchronizations of the metadata caches serving the target file in the parallel case. Synchronization occurs whenever the number of bytes of dirty metadata created since the last synchronization exceeds this limit. This field only applies to the parallel case. While it is ignored elsewhere, it can still draw a value out of bounds error. It must be consistant across all caches on any given file. By default, this field is set to 256 KB. It shouldn't be more than half the current max cache size times the min clean fraction.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Name: H5Pset_meta_block_size

Signature:

herr_t `H5Pset_meta_block_size`(*hid_t* `fapl_id`, *hsize_t* `size`)

Purpose:

Sets the minimum metadata block size.

Description:

`H5Pset_meta_block_size` sets the minimum size, in bytes, of metadata block allocations when `H5FD_FEAT_AGGREGATE_METADATA` is set by a VFL driver.

Each *raw* metadata block is initially allocated to be of the given size. Specific metadata objects (e.g., object headers, local heaps, B-trees) are then sub-allocated from this block.

The default setting is 2048 bytes, meaning that the library will attempt to aggregate metadata in at least 2K blocks in the file. Setting the value to `0` (zero) with this function will turn off metadata aggregation, even if the VFL driver attempts to use the metadata aggregation strategy.

Metadata aggregation reduces the number of small data objects in the file that would otherwise be required for metadata. The aggregated block of metadata is usually written in a single write action and always in a contiguous block, potentially significantly improving library and application performance.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t `size`	IN: Minimum size, in bytes, of metadata block allocations.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_meta_block_size_f

```
SUBROUTINE h5pset_meta_block_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! File access property list
                                         ! identifier
  INTEGER(HSIZE_T), INTENT(IN) :: size   ! Metadata block size
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_meta_block_size_f
```

History:

Release	C
1.4.0	Function introduced in this release.

Name: H5Pset_multi_type

Signature:

herr_t `H5Pset_multi_type` (*hid_t* `fapl_id`, *H5FD_mem_t* `type`)

Purpose:

Specifies type of data to be accessed via the `MULTI` driver, enabling more direct access.

Description:

`H5Pset_multi_type` sets the *type of data* property in the file access property list `fapl_id`.

This setting enables a user application to specify the type of data the application wishes to access so that the application can retrieve a file handle for low-level access to the particular member of a set of `MULTI` files in which that type of data is stored. The file handle is retrieved with a separate call to `H5Fget_vfd_handle` (or, in special circumstances, to `H5FDget_vfd_handle`; see *Virtual File Layer* and *List of VFL Functions* in *HDF5 Technical Notes*).

The type of data specified in `type` may be one of the following:

`H5FD_MEM_SUPER`	Super block data
`H5FD_MEM_BTREE`	B-tree data
`H5FD_MEM_DRAW`	Dataset raw data
`H5FD_MEM_GHEAP`	Global heap data
`H5FD_MEM_LHEAP`	Local Heap data
`H5FD_MEM_OHDR`	Object header data

This function is for use only when accessing an HDF5 file written as a set of files with the `MULTI` file driver.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
H5FD_mem_t `type`	IN: Type of data to be accessed.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**Change**
1.6.0	C function introduced in this release.

Name: H5Pset_nbit

Signature:

herr_t `H5Pset_nbit`(*hid_t* `plist_id`)

Purpose:

Sets up the use of the N-Bit filter.

Description:

`H5Pset_nbit` sets the N-Bit filter, `H5Z_FILTER_NBIT`, in the dataset creation property list `plist_id`.

The HDF5 user can create an N-Bit datatype by writing codes like:

```
hid_t nbit_datatype = H5Tcopy(H5T_STD_I32LE);
H5Tset_precision(nbit_datatype, 16);
H5Tset_offset(nbit_datatype, 4);
```

In memory, one value of the N-Bit datatype in the above example will be stored on a little-endian machine like this:

byte 3	byte 2	byte 1	byte 0
????????	????SPPP	PPPPPPPP	PPPP????

Note: S - sign bit, P - significant bit, ? - padding bit; For signed integer, the sign bit is included in the precision.

When data of the above datatype are stored on disk using N-bit filter, all padding bits are chopped off and only significant bits are stored. So, the values on disk will be something like:

1st value	2nd value	
SPPPPPPPPPPPPPPP	SPPPPPPPPPPPPPPP	...

The N-Bit filter is used effectively for compressing data of an N-Bit datatype as well as a compound and an array datatype with N-Bit fields. However, the datatype classes of the N-Bit datatype or the N-Bit field of the compound datatype or the array datatype are limited to integer or floating-point.

The N-Bit filter supports complex situations where a compound datatype contains member(s) of compound datatype or an array datatype that has compound datatype as the base type. However, it does not support the situation where an array datatype has variable-length or variable-length string as its base datatype. But the filter does support the situation where variable-length or variable-length string is a member of a compound datatype.

For all other HDF5 datatypes such as time, string, bitfield, opaque, reference, enum, and variable length, the N-Bit filter allows them to pass through like an no-op.

Like other I/O filters supported by the HDF5 library, application using the N-Bit filter must store data with chunked storage.

By nature, the N-Bit filter should not be used together with other I/O filters.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE h5pset_nbit_f(plist_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! Dataset creation property list id
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_nbit_f
```

Name: H5Pset_nlinks

Signature:

herr_t `H5Pset_nlinks(` *hid_t* `lapl_id`, *size_t* `nlinks`)

Purpose:

Sets maximum number of soft or user-defined link traversals.

Description:

`H5Pset_nlinks` sets the maximum number of soft or user-defined link traversals allowed, `nlinks`, before the library assumes it has found a cycle and aborts the traversal. This value is set in the link access property list `lapl_id`.

The limit on the number soft or user-defined link traversals is designed to terminate link traversal if one or more links form a cycle. User control is provided because some files may have legitimate paths formed of large numbers of soft or user-defined links. This property can be used to allow traversal of as many links as desired.

Parameters:

hid_t `lapl_id`	IN: File access property list identifier
size_t `nlinks`	IN: Maximum number of links to traverse

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_nlinks_f

```
SUBROUTINE h5pset_nlinks_f(lapl_id, nlinks, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: lapl_id ! File access property list identifier
  INTEGER(SIZE_T), INTENT(IN) :: nlinks ! Maximum number of links to traverse
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_nlinks_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_preserve

Signature:

herr_t `H5Pset_preserve`(*hid_t* `plist`, *hbool_t* `status`)

Purpose:

Sets the dataset transfer property list `status` to `1` (`TRUE`) or `0` (`FALSE`).

Notice:

This function is deprecated as it no longer has any effect; compound datatype field preservation is now core functionality in the HDF5 Library.

Description:

`H5Pset_preserve` sets the dataset transfer property list `status` to `1` (`TRUE`) or `0` (`FALSE`).

When reading or writing compound datatypes and the destination is partially initialized and the read/write is intended to initialize the other members, one must set this property to TRUE. Otherwise the I/O pipeline treats the destination datapoints as completely uninitialized.

Parameters:

hid_t `plist`	IN: Identifier for the dataset transfer property list.
hbool_t `status`	IN: Status of for the dataset transfer property list (TRUE/FALSE).

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_preserve_f

```
SUBROUTINE h5pset_preserve_f(prp_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id   ! Dataset transfer property
                                         ! list identifier
  LOGICAL, INTENT(IN)        :: flag     ! Status for the dataset
                                         ! transfer property list
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_preserve_f
```

History:

Release	**Fortran90**
1.6.0	The `flag` parameter has changed from *INTEGER* to *LOGICAL* to better match the C API.

Name: H5Pset_obj_track_times

Signature:

herr_t `H5Pset_obj_track_times(` *hid_t* `ocpl_id`, *hbool_t* `track_times`)

Purpose:

Sets the recording of times associated with an object.

Description:

`H5Pset_obj_track_times` sets a property in the object creation property list, `ocpl_id`, that governs the recording of times associated with an object.

If `track_times` is `TRUE`, the following times will be recorded:

Birth time	The time the object was created
Access time	The last time that metadata or raw data was read from the object
Modification time	The last time data for this object was changed (by writing raw data to a dataset or inserting, modifying, or deleting a link in a group)
Change time	The last time metadata for this object was written (by adding, modifying, or deleting an attribute on an object; extending the size of a dataset; et cetera)

If `track_times` is `FALSE`, time data will not be recorded.

Time data can be retrieved with `H5Oget_info`, which will return it in the `H5O_info_t` struct.

If times are not tracked, they will be reported as follows when queried:

```
12:00 AM UDT, Jan. 1, 1970
```

That date and time are commonly used to represent the beginning of the UNIX epoch.

Parameters:

hid_t `ocpl_id`	IN: Object creation property list identifier
hbool_t `track_times`	IN: Boolean value, `TRUE` or `FALSE`, specifying whether object times are to be tracked

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_obj_track_times_f

```
SUBROUTINE h5pset_obj_track_times_f(plist_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id
                               ! Dataset creation property
                               ! list identifier
  LOGICAL, INTENT(IN) :: flag    ! Object timestamp setting
  INTEGER, INTENT(OUT) :: hdferr ! Error code
                               ! 0 on success and -1 on failure
END SUBROUTINE h5pset_obj_track_times_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_scaleoffset

Signature:

herr_t `H5Pset_scaleoffset`(*hid_t* `plist_id`, *H5Z_SO_scale_type_t* `scale_type`, *int* `scale_factor`)

Purpose:

Sets up the use of the Scale-Offset filter.

Description:

`H5Pset_scaleoffset` sets the Scale-Offset filter, `H5Z_FILTER_SCALEOFFSET`, for a dataset.

Generally speaking, Scale-Offset compression performs a scale and/or offset operation on each data value and truncates the resulting value to a minimum number of bits (MinBits) before storing it. The current Scale-Offset filter supports integer and floating-point datatype.

For integer datatype, the parameter `scale_type` should be set to `H5Z_SO_INT (2)`. The parameter `scale_factor` denotes MinBits. If the user sets it to `H5Z_SO_INT_MINBITS_DEFAULT (0)` or less than 0, the filter will calculate MinBits. If `scale_factor` is set to a positive integer, the filter does not do any calculation and just uses the number as MinBits. However, if the user gives a MinBits that is less than what would be generated by the filter, the compression will be lossy. Also, the MinBits supplied by the user cannot exceed the number of bits to store one value of the dataset datatype.

For floating-point datatype, the filter adopts the GRiB data packing mechanism, which offers two alternate methods: E-scaling and D-scaling. Both methods are lossy compression. If the parameter `scale_type` is set to `H5Z_SO_FLOAT_DSCALE (0)`, the filter will use the D-scaling method; if it is set to `H5Z_SO_FLOAT_ESCALE (1)`, the filter will use the E-scaling method. Since only the D-scaling method is implemented, `scale_type` should be set to `H5Z_SO_FLOAT_DSCALE` or 0.

When the D-scaling method is used, the original data is "D" scaled--multiplied by 10 to the power of `scale_factor`, and the "significant" part of the value is moved to the left of the decimal point. Care should be taken in setting the decimal `scale_factor` so that the integer part will have enough precision to contain the appropriate information of the data value. For example, if `scale_factor` is set to 2, the number 104.561 will be 10456.1 after "D" scaling. The last digit 1 is not "significant" and is thrown off in the process of rounding. The user should make sure that after "D" scaling and rounding, the data values are within the range that can be represented by the integer (same size as the floating-point type).

Valid values for `scale_type` are as follows:

`H5Z_SO_FLOAT_DSCALE (0)`	Floating-point type, using variable MinBits method
`H5Z_SO_FLOAT_ESCALE (1)`	Floating-point type, using fixed MinBits method
`H5Z_SO_INT (2)`	Integer type

The meaning of `scale_factor` varies according to the value assigned to `scale_type`:

`scale_type` value	*`scale_factor` description*
`H5Z_SO_FLOAT_DSCALE`	Denotes the decimal scale factor for D-scaling and can be positive, negative or zero. This is the current implementation of the library.

`H5Z_SO_FLOAT_ESCALE`	Denotes MinBits for E-scaling and must be a positive integer. This is not currently implemented by the library.
`H5Z_SO_INT`	Denotes MinBits and it should be a positive integer or `H5Z_SO_INT_MINBITS_DEFAULT (0).` If it is less than 0, the library will reset it to 0 since it is not implemented.

Like other I/O filters supported by the HDF5 library, application using the Scale-Offset filter must store data with chunked storage.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.
H5Z_SO_scale_type_t `scale_type`	IN: Flag indicating compression method.
int `scale_factor`	IN: Parameter related to scale.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_scaleoffset_f

```
SUBROUTINE h5pset_scaleoffset_f(plist_id, scale_type, scale_factor, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! Dataset creation property list id
  INTEGER, INTENT(IN) :: scale_type      ! Flag indicating compression method
  INTEGER, INTENT(IN) :: scale_factor    ! parameter related to scale.
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_scaleoffset_f
```

Name: H5Pset_shared_mesg_index

Signature:

herr_t `H5Pset_shared_mesg_index(` *hid_t* `fcpl_id`, *unsigned* `index_num`, *unsigned* `mesg_type_flags`, *unsigned* `min_mesg_size`)

Purpose:

Configures the specified shared object header message index.

Description:

`H5Pset_shared_mesg_index` is used to configure the specified shared object header message index, setting the types of messages that may be stored in the index and the minimum size of each message.

`fcpl_id` specifies the file creation property list.

`index_num` specifies the index to be configured. `index_num` is zero-indexed, so in a file with three indexes, they will be numbered `0`, `1`, and `2`.

`mesg_type_flags` and `min_mesg_size` specify, respectively, the types and minimum size of messages that can be stored in this index.

Valid message types are as follows:

`H5O_SHMESG_NONE_FLAG`	No shared messages
`H5O_SHMESG_SDSPACE_FLAG`	Simple dataspace message
`H5O_SHMESG_DTYPE_FLAG`	Datatype message
`H5O_SHMESG_FILL_FLAG`	Fill value message
`H5O_SHMESG_PLINE_FLAG`	Filter pipeline message
`H5O_SHMESG_ATTR_FLAG`	Attribute message
`H5O_SHMESG_ALL_FLAG`	All message types; i.e., equivalent to the following: `(H5O_SHMESG_SDSPACE_FLAG | H5O_SHMESG_DTYPE_FLAG | H5O_SHMESG_FILL_FLAG | H5O_SHMESG_PLINE_FLAG | H5O_SHMESG_ATTR_FLAG)`

Parameters:

hid_t `fcpl_id`	IN: File creation property list identifier.
unsigned `index_num`	IN: Index being configured.
unsigned `mesg_type_flags`	IN: Types of messages that should be stored in this index.
unsigned `min_mesg_size`	IN: Minimum message size.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_shared_mesg_nindexes

Signature:

herr_t `H5Pset_shared_mesg_nindexes`(*hid_t* `plist_id`, *unsigned* `nindexes`)

Purpose:

Sets number of shared object header message indexes.

Description:

`H5Pset_shared_mesg_nindexes` sets the number of shared object header message indexes in the specified file creation property list.

This setting determines the number of shared object header message indexes that will be available in files created with this property list. These indexes can then be configured with `H5Pset_shared_mesg_index`.

If `nindexes` is set to `0` (zero), shared object header messages are disabled in files created with this property list.

Parameters:

hid_t `plist_id`	IN: File creation property list
unsigned `nindexes`	IN: Number of shared object header message indexes to be available in files created with this property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_shared_mesg_phase_change

Signature:

herr_t `H5Pset_shared_mesg_phase_change`(*hid_t* `fcpl_id`, *unsigned* `max_list`, *unsigned* `min_btree`)

Purpose:

Sets shared object header message storage phase change thresholds.

Description:

`H5Pset_shared_mesg_phase_change` sets threshold values for storage of shared object header message indexes in a file. These phase change thresholds determine the point at which the index storage mechanism changes from a more compact list format to a more performance-oriented B-tree format, and vice-versa.

By default, a shared object header message index is initially stored as a compact list. When the number of messages in an index exceeds the threshold value of `max_list`, storage switches to a B-tree for impoved performance. If the number of messages subsequently falls below the `min_btree` threshold, the index will revert to the list format.

If `max_compact` is set to `0` (zero), shared object header message indexes in the file will be created as B-trees and will never revert to lists.

`fcpl_id` specifies the file creation property list.

Parameters:

hid_t `fcpl_id`	IN: File creation property list identifier
unsigned `max_list`	IN: Threshold above which storage of a shared object header message index shifts from list to B-tree
unsigned `min_btree`	IN: Threshold below which storage of a shared object header message index reverts to list format

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Pset_shuffle
Signature:

herr_t `H5Pset_shuffle`(*hid_t* `plist_id`)

Purpose:

Sets up use of the shuffle filter.

Description:

`H5Pset_shuffle` sets the shuffle filter, `H5Z_FILTER_SHUFFLE`, in the dataset creation property list `plist_id`.

The shuffle filter de-interlaces a block of data by reordering the bytes. All the bytes from one consistent byte position of each data element are placed together in one block; all bytes from a second consistent byte position of each data element are placed together a second block; etc. For example, given three data elements of a 4-byte datatype stored as `012301230123`, shuffling will re-order data as `000111222333`. This can be a valuable step in an effective compression algorithm because the bytes in each byte position are often closely related to each other and putting them together can increase the compression ratio.

As implied above, the primary value of the shuffle filter lies in its coordinated use with a compression filter; it does not provide data compression when used alone. When the shuffle filter is applied to a dataset immediately prior to the use of a compression filter, the compression ratio achieved is often superior to that achieved by the use of a compression filter without the shuffle filter.

Parameters:

hid_t `plist_id`	IN: Dataset creation property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_shuffle_f

```
SUBROUTINE h5pset_shuffle_f(prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id      ! Property list identifier
  INTEGER, INTENT(OUT)       :: hdferr      ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5pset_shuffle_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Pset_sieve_buf_size

Signature:

herr_t `H5Pset_sieve_buf_size`(*hid_t* `fapl_id`, *hsize_t* `size`)

Purpose:

Sets the maximum size of the data sieve buffer.

Description:

`H5Pset_sieve_buf_size` sets `size`, the maximum size in bytes of the data sieve buffer, which is used by file drivers that are capable of using data sieving.

The data sieve buffer is used when performing I/O on datasets in the file. Using a buffer which is large enough to hold several pieces of the dataset being read in for hyperslab selections boosts performance by quite a bit.

The default value is set to 64KB, indicating that file I/O for raw data reads and writes will occur in at least 64KB blocks. Setting the value to 0 with this API function will turn off the data sieving, even if the VFL driver attempts to use that strategy.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t `size`	IN: Maximum size, in bytes, of data sieve buffer.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface: h5pset_sieve_buf_size_f

```
SUBROUTINE h5pset_sieve_buf_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! File access property list
                                         ! identifier
  INTEGER(SIZE_T), INTENT(IN) :: size    ! Sieve buffer size
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_sieve_buf_size_f
```

History:

Release	**C**
1.6.0	The `size` parameter has changed from type *hsize_t* to *size_t*.
1.4.0	Function introduced in this release.

Name: H5Pset_sizes

Signature:

herr_t `H5Pset_sizes`(*hid_t* `plist`, *size_t* `sizeof_addr`, *size_t* `sizeof_size`)

Purpose:

Sets the byte size of the offsets and lengths used to address objects in an HDF5 file.

Description:

`H5Pset_sizes` sets the byte size of the offsets and lengths used to address objects in an HDF5 file. This function is only valid for file creation property lists. Passing in a value of 0 for one of the `sizeof_...` parameters retains the current value. The default value for both values is the same as `sizeof(hsize_t)` in the library (normally 8 bytes). Valid values currently are 2, 4, 8 and 16.

Parameters:

hid_t `plist`	IN: Identifier of property list to modify.
size_t `sizeof_addr`	IN: Size of an object offset in bytes.
size_t `sizeof_size`	IN: Size of an object length in bytes.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_sizes_f

```
SUBROUTINE h5pset_sizes_f (prp_id, sizeof_addr, sizeof_size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id       ! Property list identifier
  INTEGER(SIZE_T), INTENT(IN) :: sizeof_addr ! Size of an object offset
                                             ! in bytes
  INTEGER(SIZE_T), INTENT(IN) :: sizeof_size ! Size of an object length
                                             ! in bytes
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5pset_sizes_f
```

Name: H5Pset_small_data_block_size

Signature:

herr_t `H5Pset_small_data_block_size`(*hid_t* `fapl_id`, *hsize_t* `size`)

Purpose:

Sets the size of a contiguous block reserved for small data.

Description:

`H5Pset_small_data_block_size` reserves blocks of `size` bytes for the contiguous storage of the raw data portion of *small* datasets. The HDF5 library then writes the raw data from small datasets to this reserved space, thus reducing unnecessary discontinuities within blocks of meta data and improving I/O performance.

A small data block is actually allocated the first time a qualifying small dataset is written to the file. Space for the raw data portion of this small dataset is suballocated within the small data block. The raw data from each subsequent small dataset is also written to the small data block until it is filled; additional small data blocks are allocated as required.

The HDF5 library employs an algorithm that determines whether I/O performance is likely to benefit from the use of this mechanism with each dataset as storage space is allocated in the file. A larger `size` will result in this mechanism being employed with larger datasets.

The small data block size is set as an allocation property in the file access property list identified by `fapl_id`.

Setting `size` to zero (`0`) disables the small data block mechanism.

Parameters:

hid_t `fapl_id`	IN: File access property list identifier.
hsize_t `size`	IN: Maximum size, in bytes, of the small data block. The default size is `2048`.

Returns:

Returns a non-negative value if successful; otherwise a negative value.

Fortran90 Interface: h5pset_small_data_block_size_f

```
SUBROUTINE h5pset_small_data_block_size_f(plist_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: plist_id ! File access
                                         ! property list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: size   ! Small raw data block size
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5pset_small_data_block_size_f
```

History:

Release	**C**
1.4.4	Function introduced in this release.

Name: H5Pset_sym_k

Signature:

herr_t `H5Pset_sym_k`(*hid_t* `plist`, *unsigned* `ik`, *unsigned* `lk`)

Purpose:

Sets the size of parameters used to control the symbol table nodes.

Description:

`H5Pset_sym_k` sets the size of parameters used to control the symbol table nodes. This function is only valid for file creation property lists. Passing in a value of 0 for one of the parameters retains the current value.

`ik` is one half the rank of a tree that stores a symbol table for a group. Internal nodes of the symbol table are on average 75% full. That is, the average rank of the tree is 1.5 times the value of `ik`.

`lk` is one half of the number of symbols that can be stored in a symbol table node. A symbol table node is the leaf of a symbol table tree which is used to store a group. When symbols are inserted randomly into a group, the group's symbol table nodes are 75% full on average. That is, they contain 1.5 times the number of symbols specified by `lk`.

Parameters:

hid_t `plist`	IN: File creation property list identifier.
unsigned `ik`	IN: Symbol table tree rank.
unsigned `lk`	IN: Symbol table node size.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_sym_k_f

```
SUBROUTINE h5pset_sym_k_f (prp_id, ik, lk, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
  INTEGER, INTENT(IN) :: ik            ! Symbol table tree rank
  INTEGER, INTENT(IN) :: lk            ! Symbol table node size
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5pset_sym_k_f
```

History:

Release	**C**
1.6.4	`ik` parameter type changed to *unsigned*.
1.6.0	The `ik` parameter has changed from type *int* to *unsigned*.

Name: H5Pset_szip

Signature:

herr_t `H5Pset_szip`(*hid_t* `plist`, *unsigned int* `options_mask`, *unsigned int* `pixels_per_block`)

Purpose:

Sets up use of the SZIP compression filter.

Description:

`H5Pset_szip` sets an SZIP compression filter, `H5Z_FILTER_SZIP`, for a dataset. SZIP is a compression method designed for use with scientific data.

Before proceeding, be aware that there are factors that affect your rights and ability to use SZIP compression. See the documents at SZIP Compression in HDF5 for *important information regarding terms of use and the SZIP copyright notice*, for further discussion of SZIP compression in HDF5, and for a list of SZIP-related references.

In the text below, the term *pixel* refers to an HDF5 data element. This terminology derives from SZIP compression's use with image data, where pixel referred to an image pixel.

The SZIP `bits_per_pixel` value (see **Notes**, below) is automatically set, based on the HDF5 datatype. SZIP can be used with atomic datatypes that may have size of 8, 16, 32, or 64 bits. Specifically, a dataset with a datatype that is 8-, 16-, 32-, or 64-bit signed or unsigned integer; char; or 32- or 64-bit float can be compressed with SZIP. See **Notes**, below, for further discussion of the the SZIP `bits_per_pixel` setting.

SZIP compression cannot be applied to compound datatypes, array datatypes, variable-length datatypes, enumerations, or any other user-defined datatypes. If an SZIP filter is set in a dataset creation property list used to create a dataset containing a non-allowed datatype, the call to `H5Dcreate` will fail; the conflict can be detected only when the property list is used.

SZIP options are passed in an options mask, `options_mask`, as follows.

Option	**Description** (Mutually exclusive; select one.)
`H5_SZIP_EC_OPTION_MASK`	Selects entropy coding method.
`H5_SZIP_NN_OPTION_MASK`	Selects nearest neighbor coding method.

The following guidelines can be used in determining which option to select:

◊ The entropy coding method, the EC option specified by `H5_SZIP_EC_OPTION_MASK`, is best suited for data that has been processed. The EC method works best for small numbers.

◊ The nearest neighbor coding method, the NN option specified by `H5_SZIP_NN_OPTION_MASK`, preprocesses the data then the applies EC method as above.

Other factors may affect results, but the above criteria provides a good starting point for optimizing data compression.

SZIP compresses data block by block, with a user-tunable block size. This block size is passed in the parameter `pixels_per_block` and must be even and not greater than 32, with typical values being `8`, `10`, `16`, or `32`. This parameter affects compression ratio; the more pixel values vary, the smaller this number should be to achieve better performance.

In HDF5, compression can be applied only to chunked datasets. If `pixels_per_block` is bigger than the total number of elements in a dataset chunk, `H5Pset_szip` will succeed but the subsequent call to `H5Dcreate` will fail; the conflict can be detected only when the property list is used.

To achieve optimal performance for SZIP compression, it is recommended that a chunk's fastest-changing dimension be equal to *N* times `pixels_per_block` where *N* is the maximum number of blocks per scan line allowed by the SZIP library. In the current version of SZIP, *N* is set to 128.

SZIP compression is an optional HDF5 filter. See the note below for information regarding its designed behavior, particularly under circumstances where SZIP is not available to an application.

Parameters:

hid_t `plist`	IN: Dataset creation property list identifier.
unsigned int `options_mask`	IN: A bit-mask conveying the desired SZIP options. Valid values are `H5_SZIP_EC_OPTION_MASK` and `H5_SZIP_NN_OPTION_MASK`.
unsigned int `pixels_per_block`	IN: The number of pixels or data elements in each data block.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Notes for Users Familiar with SZIP in Other Contexts:

The following notes are of interest primarily to those who have used SZIP compression outside of the HDF5 context.

In non-HDF5 applications, SZIP typically requires that the user application supply additional parameters:

◊ `pixels_in_object`, the number of pixels in the object to be compressed
◊ `bits_per_pixel`, the number of bits per pixel
◊ `pixels_per_scanline`, the number of pixels per scan line

These values need not be independently supplied in the HDF5 environment as they are derived from the datatype and dataspace, which are already known. In particular, HDF5 sets `pixels_in_object` to the number of elements in a chunk and `bits_per_pixel` to the size of the element or pixel datatype. The following algorithm is used to set `pixels_per_scanline`:

◊ If the size of a chunk's fastest-changing dimension, *size*, is greater than 4K, set `pixels_per_scanline` to 128 times `pixels_per_block`.
◊ If *size* is less than 4K but greater than `pixels_per_block`, set `pixels_per_scanline` to the minimum of *size* and 128 times `pixels_per_block`.
◊ If *size* is less than `pixels_per_block` but greater than the number elements in the chunk, set `pixels_per_scanline` to the minimum of the number elements in the chunk and 128 times `pixels_per_block`.

The HDF5 datatype may have precision that is less than the full size of the data element, e.g., an 11-bit integer can be defined using `H5Tset_precision`. To a certain extent, SZIP can take advantage of the precision of the datatype to improve compression:

- ◊ If the HDF5 datatype size is 24-bit or less and the offset of the bits in the HDF5 datatype is zero (see `H5Tset_offset` or `H5Tget_offset`), the data is the in lowest N bits of the data element. In this case, the SZIP `bits_per_pixel` is set to the precision of the HDF5 datatype.
- ◊ If the offset is not zero, the SZIP `bits_per_pixel` will be set to the number of bits in the full size of the data element.
- ◊ If the HDF5 datatype precision is 25-bit to 32-bit, the SZIP `bits_per_pixel` will be set to 32.
- ◊ If the HDF5 datatype precision is 33-bit to 64-bit, the SZIP `bits_per_pixel` will be set to 64.

HDF5 always modifies the options mask provided by the user to set up usage of `RAW_OPTION_MASK`, `ALLOW_K13_OPTION_MASK`, and one of `LSB_OPTION_MASK` or `MSB_OPTION_MASK`, depending on endianness of the datatype.

Fortran90 Interface: h5pset_szip_f

```
SUBROUTINE h5pset_szip_f(prp_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id
                                   ! Dataset creation property list identifier
  INTEGER, INTENT(IN) :: options_mask
                                   ! A bit-mask conveying the desired
                                   ! SZIP options
                                   ! Current valid values in Fortran are:
                                   !    H5_SZIP_EC_OM_F
                                   !    H5_SZIP_NN_OM_F
  INTEGER, INTENT(IN) :: pixels_per_block
                                   ! The number of pixels or data elements
                                   ! in each data block
  INTEGER, INTENT(OUT)  :: hdferr  ! Error code
                                   ! 0 on success and -1 on failure
END SUBROUTINE h5pset_szip_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Last modified: 2 April 2009

Name: H5Pset_type_conv_cb

Signature:

herr_t `H5Pset_type_conv_cb`(*hid_t* `plist`, *H5T_conv_except_func_t* `func`, *void* *`op_data`)

Purpose:

Sets user-defined datatype conversion callback function.

Description:

`H5Pset_type_conv_cb` sets the user-defined datatype conversion callback function `func` in the dataset transfer property list `plist`.

The parameter `op_data` is a pointer to user-defined input data for the callback function and will be passed through to the callback function.

The callback function `func` defines the actions an application is to take when there is an exception during datatype conversion. The function prototype is as follows:

`typedef` *H5T_conv_ret_t* (`H5T_conv_except_func_t`) (*H5T_conv_except_t* `except_type`, *hid_t* *`src_id`, *hid_t* *`dst_id`, *void* *`src_buf`, *void* *`dst_buf`, *void* *`op_data`)

where `except_type` indicates what kind of exception has happened, `src_id` and `dst_id` are the source and destination datatype identifiers, `src_buf` and `dst_buf` are the source and destination data buffer, and `op_data` is the required input data for this callback function.

Valid values for `except_type` are as follows:

`H5T_CONV_EXCEPT_RANGE_HI`
: Source value is positive and is too big to the destination. Overflow happens.

`H5T_CONV_EXCEPT_RANGE_LOW`
: Source value is negative and its magnitude is too big to the destination. Overflow happens.

`H5T_CONV_EXCEPT_TRUNCATE`
: Source is floating-point type and destination is integer. The floating-point number has fractional part.

`H5T_CONV_EXCEPT_PRECISION`
: Source is integer and destination is floating-point type. The mantissa of floating-point type is not big enough to hold all the digits of the integer.

`H5T_CONV_EXCEPT_PINF`
: Source is floating-point type and the value is positive infinity.

`H5T_CONV_EXCEPT_NINF`
: Source is floating-point type and the value is negative infinity.

`H5T_CONV_EXCEPT_NAN`

Source is floating-point type and the value is NaN (not a number, including QNaN and SNaN).

Valid callback function return values are `H5T_CONV_ABORT`, `H5T_CONV_UNHANDLED` and `H5T_CONV_HANDLED`.

Parameters:

hid_t `plist`	IN: Dataset transfer property list identifier.
H5T_conv_except_func_t `func`	IN: User-defined type conversion callback function.< /td>
void *`op_data`	IN: User-defined input data for the callback function.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

Name: H5Pset_userblock
Signature:
herr_t `H5Pset_userblock`(*hid_t* `plist`, *hsize_t* `size`)
Purpose:
Sets user block size.
Description:
`H5Pset_userblock` sets the user block size of a file creation property list. The default user block size is 0; it may be set to any power of 2 equal to 512 or greater (512, 1024, 2048, etc.).
Parameters:

hid_t `plist`	IN: Identifier of property list to modify.
hsize_t `size`	IN: Size of the user-block in bytes.

Returns:
Returns a non-negative value if successful; otherwise returns a negative value.
Fortran90 Interface: h5pset_userblock_f
```
SUBROUTINE h5pset_userblock_f (prp_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: prp_id  ! Property list identifier
  INTEGER(HSIZE_T), INTENT(IN) :: size  ! Size of the user-block in bytes
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_userblock_f
```

Name: H5Pset_vlen_mem_manager
Signature:
herr_t `H5Pset_vlen_mem_manager`(*hid_t* `plist`, *H5MM_allocate_t* `alloc`, *void* *`alloc_info`, *H5MM_free_t* `free`, *void* *`free_info`)

Purpose:
Sets the memory manager for variable-length datatype allocation in `H5Dread` and `H5Dvlen_reclaim`.

Description:
`H5Pset_vlen_mem_manager` sets the memory manager for variable-length datatype allocation in `H5Dread` and free in `H5Dvlen_reclaim`.

The `alloc` and `free` parameters identify the memory management routines to be used. If the user has defined custom memory management routines, `alloc` and/or `free` should be set to make those routine calls (i.e., the name of the routine is used as the value of the parameter); if the user prefers to use the system's `malloc` and/or `free`, the `alloc` and `free` parameters, respectively, should be set to `NULL`

The prototypes for these user-defined functions would appear as follows:
typedef void *(*`H5MM_allocate_t`)(*size_t* `size`, *void* *`alloc_info`) ;
typedef void (*`H5MM_free_t`)(*void* *`mem`, *void* *`free_info`) ;
The `alloc_info` and `free_info` parameters can be used to pass along any required information to the user's memory management routines.

In summary, if the user has defined custom memory management routines, the name(s) of the routines are passed in the `alloc` and `free` parameters and the custom routines' parameters are passed in the `alloc_info` and `free_info` parameters. If the user wishes to use the system `malloc` and `free` functions, the `alloc` and/or `free` parameters are set to `NULL` and the `alloc_info` and `free_info` parameters are ignored.

Parameters:

hid_t `plist`	IN: Identifier for the dataset transfer property list.
H5MM_allocate_t `alloc`	IN: User's allocate routine, or `NULL` for system `malloc`.
void *`alloc_info`	IN: Extra parameter for user's allocation routine. Contents are ignored if preceding parameter is `NULL`.
H5MM_free_t `free`	IN: User's free routine, or `NULL` for system `free`.
void *`free_info`	IN: Extra parameter for user's free routine. Contents are ignored if preceding parameter is `NULL`.

Returns:
Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:
None.

Name: H5Punregister

Signature:

herr_t `H5Punregister(` *H5P_class_t* `class`, *const char* `*name`)

Purpose:

Removes a property from a property list class.

Description:

`H5Punregister` removes a property from a property list class.

Future property lists created of that class will not contain this property; existing property lists containing this property are not affected.

Parameters:

H5P_class_t `class`	IN: Property list class from which to remove permanent property
const char `*name`	IN: Name of property to remove

Returns:

Success: a non-negative value

Failure: a negative value

Fortran90 Interface: h5punregister_f

```
SUBROUTINE h5punregister_f(class, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: class  ! Property list class identifier
  CHARACTER(LEN=*), INTENT(IN) :: name ! Name of property to remove
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5punregister_f
```

H5R: Reference Interface

Reference API Functions

The Reference interface allows the user to create references to specific objects and data regions in an HDF5 file. In the following lists, italic type indicates a configurable macro.

The C Interfaces:

- H5Rcreate
- H5Rdereference
- *H5Rget_obj_type*
- H5Rget_obj_type1 *
- H5Rget_obj_type2
- H5Rget_region
- H5Rget_name

** Use of this function is deprecated in Release 1.8.0.*

Alphabetical Listing

- H5Rcreate
- H5Rdereference
- H5Rget_name
- *H5Rget_obj_type*
- H5Rget_obj_type1 *
- H5Rget_obj_type2
- H5Rget_region

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5rcreate_f
- h5rdereference_f
- h5rget_region_f
- h5rget_object_type_f
- h5rget_name_f

Name: H5Rcreate
Signature:

herr_t `H5Rcreate`(*void* *`ref`, *hid_t* `loc_id`, *const char* *`name`, *H5R_type_t* `ref_type`, *hid_t* `space_id`)

Purpose:

Creates a reference.

Description:

`H5Rcreate` creates the reference, `ref`, of the type specified in `ref_type`, pointing to the object `name` located at `loc_id`.

The HDF5 library maps the *void* type specified above for `ref` to the type specified in `ref_type`, which will be one of those appearing in the first column of the following table. The second column of the table lists the HDF5 constant associated with each reference type.

hdset_reg_ref_t	`H5R_DATASET_REGION`	Dataset region reference
hobj_ref_t	`H5R_OBJECT`	Object reference

The parameters `loc_id` and `name` are used to locate the object.

The parameter `space_id` identifies the dataset region that a dataset region reference points to. This parameter is used only with dataset region references and should be set to `-1` if the reference is an object reference, `H5R_OBJECT`.

Parameters:

void *`ref`	OUT: Reference created by the function call.
hid_t `loc_id`	IN: Location identifier used to locate the object being pointed to.
const char *`name`	IN: Name of object at location `loc_id`.
H5R_type_t `ref_type`	IN: Type of reference.
hid_t `space_id`	IN: Dataspace identifier with selection. Used only for dataset region references; pass as `-1` if reference is an object reference, i.e., of type `H5R_OBJECT`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5rcreate_f

To create an object reference

```
SUBROUTINE h5rcreate_f(loc_id, name, ref, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id     ! Location identifier
  CHARACTER(LEN=*), INTENT(IN) :: name     ! Name of the object at location
                                           ! specified by loc_id identifier
  TYPE(hobj_ref_t_f), INTENT(OUT) :: ref   ! Object reference
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure

END SUBROUTINE h5rcreate_f
```

To create a region reference

```
SUBROUTINE h5rcreate_f(loc_id, name, space_id, ref, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id        ! Location identifier
  CHARACTER(LEN=*), INTENT(IN) :: name        ! Name of the dataset at location
                                              ! specified by loc_id identifier
  INTEGER(HID_T), INTENT(IN) :: space_id      ! Dataset's dataspace identifier
  TYPE(hdset_reg_ref_t_f), INTENT(OUT) :: ref ! Dataset region reference
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure

END SUBROUTINE h5rcreate_f
```

Name: H5Rdereference

Signature:

hid_t `H5Rdereference`(*hid_t* `obj_id`, *H5R_type_t* `ref_type`, *void* `*ref`)

Purpose:

Opens the HDF5 object referenced.

Description:

Given a reference, `ref`, to an object or a region in an object, `H5Rdereference` opens that object and returns an identifier.

The parameter `obj_id` must be a valid identifier for an object in the HDF5 file containing the referenced object, including the file identifier.

The parameter `ref_type` specifies the reference type of the reference `ref`. `ref_type` may contain either of the following values:

◊ `H5R_OBJECT` (`0`)
◊ `H5R_DATASET_REGION` (`1`)

Parameters:

hid_t `obj_id`	IN: Valid identifier for the file containing the referenced object or any object in that file.
H5R_type_t `ref_type`	IN: The reference type of `ref`.
void `*ref`	IN: Reference to open.

Returns:

Returns identifier of referenced object if successful; otherwise returns a negative value.

Fortran90 Interface: h5rdereference_f

To dereference an object

```
SUBROUTINE h5rdereference_f(obj_id, ref, ref_obj_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id      ! Valid identifier
                                            ! in file
  TYPE(hobj_ref_t_f), INTENT(IN) :: ref     ! Object reference
  INTEGER(HID_T), INTENT(OUT) :: ref_obj_id ! Identifier of
                                            ! referenced object
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure

END SUBROUTINE h5rdereference_f
```

To dereference a region

```
SUBROUTINE h5rdereference_f(obj_id, ref, ref_obj_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id       ! Valid identifier
                                             ! in file
  TYPE(hdset_reg_ref_t_f), INTENT(IN) :: ref ! Object reference
  INTEGER(HID_T), INTENT(OUT) :: ref_obj_id  ! Identifier of
                                             ! referenced object
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure

END SUBROUTINE h5rdereference_f
```

Name: H5Rget_obj_type

Signature:

H5G_obj_t `H5Rget_obj_type`(*hid_t* `loc_id`, *H5R_type_t* `ref_type`, *void* `*ref`)
herr_t `H5Rget_obj_type`(*hid_t* `loc_id`, *H5R_type_t* `ref_type`, *void* `*ref`, *H5O_type_t* `*obj_type`)

Purpose:

Retrieves the type of object that an object reference points to.

Description:

`H5Rget_obj_type` is a macro that is mapped to either `H5Rget_obj_type1` or `H5Rget_obj_type2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Rget_obj_type` is mapped to the most recent version of the function, currently `H5Rget_obj_type2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Rget_obj_type` will be mapped to `H5Rget_obj_type1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Rget_obj_type` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Rget_obj_type2`
Enable deprecated symbols	`H5Rget_obj_type2`
Disable deprecated symbols	`H5Rget_obj_type2`
Emulate Release 1.6 interface	`H5Rget_obj_type1`
<u>Function-level macros</u>	
`H5Rget_obj_type_vers = 2`	`H5Rget_obj_type2`
`H5Rget_obj_type_vers = 1`	`H5Rget_obj_type1`

Fortran90 Interface: h5rget_object_type_f

```
SUBROUTINE h5rget_object_type_f(dset_id, ref, obj_type, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id   ! Dataset identifier
  TYPE(hobj_ref_t_f), INTENT(IN) :: ref   ! Object reference
  INTEGER, INTENT(OUT) :: obj_type        ! Object type
                                          !     H5G_UNKNOWN_F (-1)
                                          !     H5G_LINK_F      0
                                          !     H5G_GROUP_F     1
                                          !     H5G_DATASET_F   2
                                          !     H5G_TYPE_F      3
```

```
    INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                            ! 0 on success and -1 on failure

  END SUBROUTINE h5rget_object_type_f
```

History:

Release	**C**
1.8.0	The function `H5Rget_obj_type` renamed to `H5Rget_obj_type1` and deprecated in this release. The macro `H5Rget_obj_type` and the function `H5Rget_obj_type2` introduced in this release.

Name: H5Rget_obj_type1

Signature:

H5G_obj_t `H5Rget_obj_type1`(*hid_t* `loc_id`, *H5R_type_t* `ref_type`, *void* `*ref`)

Purpose:

Retrieves the type of object that an object reference points to.

Notice:

This function has been renamed from `H5Rget_obj_type` and is deprecated in favor of the macro `H5Rget_obj_type` or the function `H5Rget_obj_type2`.

Description:

Given an object reference, `ref`, `H5Rget_obj_type1` returns the type of the referenced object.

A *reference type* is the type of reference, either an object reference or a dataset region reference. An *object reference* points to an HDF5 object while a *dataset region reference* points to a defined region within a dataset.

The *referenced object* is the object the reference points to. The *referenced object type*, or the type of the referenced object, is the type of the object that the reference points to.

The location identifier, `loc_id`, is the identifier for either the dataset containing the object reference or the group containing that dataset.

Valid reference types, to pass in as `ref_type`, include the following:

`H5R_OBJECT`	Object reference
`H5R_DATASET_REGION`	Dataset region reference

If the application does not already know the object reference type, that can be determined with three preliminary calls:

◊ Call `H5Dget_type` on the dataset containing the reference to get a datatype identifier for the dataset's datatype.
◊ Using that datatype identifier, `H5Tget_class` returns a datatype class.
◊ If the datatype class is `H5T_REFERENCE`, `H5Tequal` can then be used to determine whether the reference's datatype is `H5T_STD_REF_OBJ` or `H5T_STD_REF_DSETREG`:
  · If the datatype is `H5T_STD_REF_OBJ`, the reference object type is `H5R_OBJECT`.
  · If the datatype is `H5T_STD_REF_DSETREG`, the reference object type is `H5R_DATASET_REGION`.

When the function completes successfully, it returns one of the following valid object type values (defined in `H5Gpublic.h`):

`H5G_LINK`	Object is a symbolic link.
`H5G_GROUP`	Object is a group.
`H5G_DATASET`	Object is a dataset.
`H5G_TYPE`	Object is a named datatype.

Parameters:

hid_t `loc_id`	IN: The dataset containing the reference object or the group containing that dataset.
H5R_type_t `ref_type`	IN: Type of reference to query.
void `*ref`	IN: Reference to query.

Returns:

Returns a valid object type if successful; otherwise returns `H5G_UNKNOWN`.

Fortran90 Interface: h5rget_object_type_f

See the `H5Rget_obj_type` macro description.

History:

Release	**C**
1.6.0	Function introduced in this release.
1.8.0	Function `H5Rget_obj_type` renamed to `H5Rget_obj_type1` and deprecated in this release.

Name: H5Rget_obj_type2

Signature:

herr_t `H5Rget_obj_type2`(*hid_t* `loc_id`, *H5R_type_t* `ref_type`, *void* `*ref`, *H5O_type_t* `*obj_type`)

Purpose:

Retrieves the type of object that an object reference points to.

Description:

Given an object reference, `ref`, `H5Rget_obj_type2` retrieves the type of the referenced object in `obj_type`.

A *reference type* is the type of reference, either an object reference or a dataset region reference. An *object reference* points to an HDF5 object while a *dataset region reference* points to a defined region within a dataset.

The *referenced object* is the object the reference points to. The *referenced object type*, or the type of the referenced object, is the type of the object that the reference points to.

The location identifier, `loc_id`, is the identifier for either the dataset containing the object reference or the group containing that dataset.

Valid reference types, to pass in as `ref_type`, include the following:

`H5R_OBJECT`	Object reference
`H5R_DATASET_REGION`	Dataset region reference

If the application does not already know the object reference type, that can be determined with three preliminary calls:

◊ Call `H5Dget_type` on the dataset containing the reference to get a datatype identifier for the dataset's datatype.
◊ Using that datatype identifier, `H5Tget_class` returns a datatype class.
◊ If the datatype class is `H5T_REFERENCE`, `H5Tequal` can then be used to determine whether the reference's datatype is `H5T_STD_REF_OBJ` or `H5T_STD_REF_DSETREG`:
  · If the datatype is `H5T_STD_REF_OBJ`, the reference object type is `H5R_OBJECT`.
  · If the datatype is `H5T_STD_REF_DSETREG`, the reference object type is `H5R_DATASET_REGION`.

When the function completes successfully, it returns one of the following valid object type values (defined in `H5Opublic.h`):

`H5O_TYPE_GROUP`	Object is a group.
`H5O_TYPE_DATASET`	Object is a dataset.
`H5O_TYPE_NAMED_DATATYPE`	Object is a named datatype.

Parameters:

hid_t `loc_id`	IN: The dataset containing the reference object or the group containing that dataset.
H5R_type_t `ref_type`	IN: Type of reference to query.
void `*ref`	IN: Reference to query.
H5O_type_t `*obj_type`	OUT: Type of referenced object.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5rget_object_type_f

See the `H5Rget_obj_type` macro description.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Rget_name
Signature:

ssize_t `H5Rget_name`(*hid_t* `loc_id`, *H5R_type_t* `ref_type`, *void* *`ref`, *char* *`name`, *size_t* `size`)

Purpose:

Retrieves a name of a referenced object.

Description:

`H5Rget_name` retrieves a name for the object identified by `ref`.

`loc_id` is the identifier for the dataset containing the reference or for the group containing that dataset.

`H5R_type_t` is the reference type of `ref`. Valid values include the following:

`H5R_OBJECT`	Object reference
`H5R_DATASET_REGION`	Dataset region reference

`ref` is the reference for which the target object's name is sought.

If `ref` is an object reference, `name` will be returned with the name of the referenced object. If `ref` is a dataset region reference, `name` will contain the name of the object containing the referenced region.

Up to `size` characters of the name are returned in `name`; additional characters, if any, are not returned to the user application.

If the length of the name, which determines the required value of `size`, is unknown, a preliminary `H5Rget_name` call can be made. The return value of this call will be the size of the object name. That value can then be assigned to `size` for a second `H5Rget_name` call, which will retrieve the actual name.

If there is no name associated with the object identifier or if the name is `NULL`, `H5Rget_name` returns `0` (zero).

Note that an object in an HDF5 file may have multiple paths if there are multiple links pointing to it. This function may return any one of these paths.

Parameters:

hid_t `loc_id`	IN: Identifier for the dataset containing the reference or for the group that dataset is in.
H5R_type_t `ref_type`	IN: Type of reference.
void *`ref`	IN: An object or dataset region reference.
char *`name`	OUT: A name associated with the referenced object or dataset region.
size_t `size`	IN: The size of the `name` buffer.

Returns:

Returns the length of the name if successful, returning `0` (zero) if no name is associated with the identifier. Otherwise returns a negative value.

Fortran90 Interface: h5rget_name_object_f

To get name of an object reference

```
SUBROUTINE h5rget_name_object_f(loc_id,  ref, name, hdferr, size)

  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id
                        ! Identifier for the dataset containing the reference
                        ! or for the group that dataset is in.
  TYPE(hobj_ref_t_f), INTENT(IN) :: ref
                        ! Object reference
  INTEGER(SIZE_T), OPTIONAL, INTENT(OUT) :: size
                        ! The size of the name buffer,
                        ! returning 0 (zero) if no name is associated with the
                        ! identifier
  CHARACTER(LEN=*), INTENT(OUT) :: name
                        ! A name associated with the referenced object or
                        ! dataset region.
  INTEGER, INTENT(OUT) :: hdferr
                        ! Error code
                        ! 0 on success and -1 on failure
  INTEGER(HADDR_T) :: ref_f
                        ! Local buffer to pass reference
END SUBROUTINE h5rget_name_object_f
```

To get name of a region reference

```
SUBROUTINE h5rget_name_region_f(loc_id, ref, name, hdferr, size)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id  ! Identifier for the dataset containing
                                        ! the reference
                                        ! or for the group that dataset is in.
  TYPE(hdset_reg_ref_t_f), INTENT(IN) :: ref
                                        ! Object reference
  INTEGER(SIZE_T), OPTIONAL, INTENT(OUT) :: size
                                        ! The size of the name buffer,
                                        ! returning 0 (zero) if no name is
                                        ! associated with the identifier.
  CHARACTER(LEN=*), INTENT(OUT) :: name ! A name associated with the
                                        ! referenced object or dataset region.
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5rget_name_region_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Last modified: 30 April 2009

Name: H5Rget_region

Signature:

hid_t `H5Rget_region`(*hid_t* `loc_id`, *H5R_type_t* `ref_type`, *void* `*ref`)

Purpose:

Sets up a dataspace and selection as specified by a region reference.

Description:

`H5Rget_region` creates a copy of the dataspace of the dataset pointed to by a region reference, `ref`, and defines a selection matching the selection pointed to by `ref` within the dataspace copy.

`loc_id` is used to identify the file containing the referenced region; it can be a file identifier or an identifier for any object in the file.

The parameter `ref_type` specifies the reference type of `ref` and must contain the following value:

◊ `H5R_DATASET_REGION` (1)

Parameters:

hid_t `loc_id`	IN: File identifier or identifier for any object in the file containing the referenced region
H5R_type_t `ref_type`	IN: Reference type of `ref`, which must be `H5R_DATASET_REGION`
void `*ref`	IN: Region reference to open

Returns:

Returns a valid dataspace identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5rget_region_f

```
SUBROUTINE h5rget_region_f(obj_id, ref, space_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id        ! Object identifier
  TYPE(hdset_reg_ref_t_f), INTENT(IN) :: ref  ! Dataset region reference
  INTEGER(HID_T), INTENT(OUT) :: space_id     ! Space identifier
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure

END SUBROUTINE h5rget_region_f
```

H5S: Dataspace Interface

Dataspace Object API Functions

These functions create and manipulate the dataspace in which to store the elements of a dataset.

The C Interfaces:

- H5Screate
- H5Scopy
- H5Sclose
- H5Sdecode
- H5Sencode
- H5Screate_simple
- H5Sis_simple
- H5Soffset_simple
- H5Sget_simple_extent_dims
- H5Sget_simple_extent_ndims
- H5Sget_simple_extent_npoints
- H5Sget_simple_extent_type
- H5Sextent_copy
- H5Sextent_equal
- H5Sset_extent_simple
- H5Sset_extent_none
- H5Sget_select_type
- H5Sget_select_npoints
- H5Sget_select_hyper_nblocks
- H5Sget_select_hyper_blocklist
- H5Sget_select_elem_npoints
- H5Sget_select_elem_pointlist
- H5Sget_select_bounds
- H5Sselect_elements
- H5Sselect_all
- H5Sselect_none
- H5Sselect_valid
- H5Sselect_hyperslab

Alphabetical Listing

- H5Sclose
- H5Scopy
- H5Screate
- H5Screate_simple
- H5Sdecode
- H5Sencode
- H5Sextent_copy
- H5Sextent_equal
- H5Sget_select_bounds
- H5Sget_select_elem_npoints
- H5Sget_select_elem_pointlist
- H5Sget_select_hyper_blocklist
- H5Sget_select_hyper_nblocks
- H5Sget_select_npoints
- H5Sget_select_type
- H5Sget_simple_extent_dims
- H5Sget_simple_extent_ndims
- H5Sget_simple_extent_npoints
- H5Sget_simple_extent_type
- H5Sis_simple
- H5Soffset_simple
- H5Sselect_all
- H5Sselect_elements
- H5Sselect_hyperslab
- H5Sselect_none
- H5Sselect_valid
- H5Sset_extent_none
- H5Sset_extent_simple

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5screate_f
- h5scopy_f
- h5sclose_f
- H5Sdecode_f
- H5Sencode_f
- h5screate_simple_f
- h5sis_simple_f
- h5soffset_simple_f
- h5sget_simple_extent_dims_f
- h5sget_simple_extent_ndims_f
- h5sget_simple_extent_npoints_f
- h5sget_simple_extent_type_f
- h5sextent_copy_f
- H5Sextent_equal_f
- h5sset_extent_simple_f
- h5sset_extent_none_f
- h5sget_select_type_f
- h5sget_select_npoints_f
- h5sget_select_hyper_nblocks_f
- h5sget_select_hyper_blocklist_f
- h5sget_select_elem_npoints_f
- h5sget_select_elem_pointlist_f
- h5sselect_elements_f
- h5sselect_all_f
- h5sselect_none_f
- h5sselect_valid_f
- h5sselect_hyperslab_f

Name: H5Sclose

Signature:

herr_t `H5Sclose`(*hid_t* `space_id`)

Purpose:

Releases and terminates access to a dataspace.

Description:

`H5Sclose` releases a dataspace. Further access through the dataspace identifier is illegal. Failure to release a dataspace with this call will result in resource leaks.

Parameters:

hid_t `space_id`	Identifier of dataspace to release.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sclose_f

```
SUBROUTINE h5sclose_f(space_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id  ! Dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5sclose_f
```

Name: H5Scopy

Signature:

hid_t `H5Scopy`(*hid_t* `space_id`)

Purpose:

Creates an exact copy of a dataspace.

Description:

`H5Scopy` creates a new dataspace which is an exact copy of the dataspace identified by `space_id`. The dataspace identifier returned from this function should be released with `H5Sclose` or resource leaks will occur.

Parameters:

hid_t `space_id`	Identifier of dataspace to copy.

Returns:

Returns a dataspace identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5scopy_f

```
SUBROUTINE h5scopy_f(space_id, new_space_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id      ! Dataspace identifier
  INTEGER(HID_T), INTENT(OUT) :: new_space_id ! Identifier of dataspace copy
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
                                              ! 0 on success and -1 on failure
END SUBROUTINE h5scopy_f
```

Last modified: 7 April 2009

Name: H5Screate

Signature:

hid_t `H5Screate`(*H5S_class_t* `type`)

Purpose:

Creates a new dataspace of a specified type.

Description:

`H5Screate` creates a new dataspace of a particular `type`. Currently supported types are as follows:

`H5S_SCALAR`
`H5S_SIMPLE`
`H5S_NULL`

Further dataspace types may be added later.

A *scalar dataspace*, `H5S_SCALAR`, has a single element, though that element may be of a complex datatype, such as a compound or array datatype. By convention, the rank of a scalar dataspace is always `0` (zero); think of it geometrically as a single, dimensionless point, though that point can be complex.

A *simple dataspace*, `H5S_SIMPLE`, consists of a regular array of elements.

A *null dataspace*, `H5S_NULL`, has no data elements.

Parameters:

H5S_class_t `type`	The type of dataspace to be created.

Returns:

Returns a dataspace identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5screate_f

```
SUBROUTINE h5screate_f(classtype, space_id, hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: classtype        ! The type of the dataspace
                                          ! to be created. Possible values
                                          ! are:
                                          !    H5S_SCALAR_F
                                          !    H5S_SIMPLE_F
                                          !    H5S_NULL_F (Not yet implemented)
  INTEGER(HID_T), INTENT(OUT) :: space_id ! Dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5screate_f
```

Last modified: 13 April 2009

Name: H5Screate_simple

Signature:

hid_t `H5Screate_simple`(*int* `rank`, *const hsize_t* * `dims`, *const hsize_t* * `maxdims`)

Purpose:

Creates a new simple dataspace and opens it for access.

Description:

`H5Screate_simple` creates a new simple dataspace and opens it for access, returning a dataset identifier.

`rank` is the number of dimensions used in the dataspace.

`dims` is a one-dimensional array of size `rank` specifying the size of each dimension of the dataset. `maxdims` is an array of the same size specifying the upper limit on the size of each dimension. `maxdims` may be the null pointer, in which case the upper limit is the same as `dims`.

If an element of `maxdims` is `H5S_UNLIMITED`, the maximum size of the corresponding dimension is unlimited. Otherwise, no element of `maxdims` should be smaller than the corresponding element of `dims`.

Note that any dataset with an unlimited dimension must also be chunked; see `H5Pset_chunk`.

The dataspace identifier returned from this function must be released with `H5Sclose` or resource leaks will occur.

Parameters:

int `rank`	Number of dimensions of dataspace.
const hsize_t * `dims`	An array of the size of each dimension.
const hsize_t * `maxdims`	An array of the maximum size of each dimension.

Returns:

Returns a dataspace identifier if successful; otherwise returns a negative value.

See Also:

`H5Pset_chunk`

Fortran90 Interface: h5screate_simple_f

```
SUBROUTINE h5screate_simple_f(rank, dims, space_id, hdferr, maxdims)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: rank             ! Number of dataspace dimensions
  INTEGER(HSIZE_T), INTENT(IN) :: dims(*) ! Array with the dimension sizes
  INTEGER(HID_T), INTENT(OUT) :: space_id ! Dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
  INTEGER(HSIZE_T), OPTIONAL, INTENT(IN) :: maxdims(*)
                                          ! Array with the maximum
                                          ! dimension sizes
END SUBROUTINE h5screate_simple_f
```

Name: H5Sdecode
Signature:
hid_t `H5Sdecode` (*unsigned char* `*buf`)
Purpose:
Decode a binary object description of data space and return a new object handle.
Description:
Given an object description of data space in binary in a buffer, `H5Sdecode` reconstructs the HDF5 data type object and returns a new object handle for it. The binary description of the object is encoded by `H5Sencode`. User is responsible for passing in the right buffer. The types of data space we address in this function are null, scalar, and simple space. For simple data space, the information of selection, for example, hyperslab selection, is also encoded and decoded. Complex data space has not been implemented in the library.
Parameters:

unsigned char `*buf`	IN: Buffer for the data space object to be decoded.

Returns:
Returns an object ID(non-negative) if successful; otherwise returns a negative value.
Fortran90 Interface: h5sdecode_f

```
SUBROUTINE h5sdecode_f(buf, obj_id, hdferr)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: buf   ! Buffer of data space object to
                                        ! be decoded.
  INTEGER(HID_T), INTENT(OUT) :: obj_id ! Object ID
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5sdecode_f
```

Name: H5Sencode

Signature:

herr_t `H5Sencode`(*hid_t* `obj_id`, *unsigned char* *`buf`, *size_t* *`nalloc`)

Purpose:

Encode a data space object description into a binary buffer.

Description:

Given the data space ID, `H5Sencode` converts a data space description into binary form in a buffer. Using this binary form in the buffer, a data space object can be reconstructed using `H5Sdecode` to return a new object handle(`hid_t`) for this data space.

A preliminary `H5Sencode` call can be made to find out the size of the buffer needed. This value is returned as `nalloc`. That value can then be assigned to `nalloc` for a second `H5Sencode` call, which will retrieve the actual encoded object.

If the library finds out `nalloc` is not big enough for the object, it simply returns the size of the buffer needed through `nalloc` without encoding the provided buffer.

The types of data space we address in this function are null, scalar, and simple space. For simple data space, the information of selection, for example, hyperslab selection, is also encoded and decoded. Complex data space has not been implemented in the library.

Parameters:

hid_t `obj_id`	IN: Identifier of the object to be encoded.
unsigned char *`buf`	IN/OUT: Buffer for the object to be encoded into. If the provided buffer is NULL, only the size of buffer needed is returned through `nalloc`.
size_t *`nalloc`	IN: The size of the allocated buffer. OUT: The size of the buffer needed.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sencode_f

```
SUBROUTINE h5sencode_f(obj_id, buf, nalloc, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id ! Identifier of the object to be encoded
  CHARACTER(LEN=*), INTENT(OUT) :: buf ! Buffer of object to be encoded into
  INTEGER(SIZE_T), INTENT(INOUT) :: nalloc
                                     ! Size of the allocated buffer
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                     ! 0 on success and -1 on failure
END SUBROUTINE h5sencode_f
```

Name: H5Sextent_copy

Signature:

herr_t `H5Sextent_copy`(*hid_t* `dest_space_id`, *hid_t* `source_space_id`)

Purpose:

Copies the extent of a dataspace.

Description:

`H5Sextent_copy` copies the extent from `source_space_id` to `dest_space_id`. This action may change the type of the dataspace.

Parameters:

hid_t `dest_space_id`	IN: The identifier for the dataspace to which the extent is copied.
hid_t `source_space_id`	IN: The identifier for the dataspace from which the extent is copied.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sextent_copy_f

```
SUBROUTINE h5sextent_copy_f(dest_space_id, source_space_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dest_space_id   ! Identifier of destination
                                                ! dataspace
  INTEGER(HID_T), INTENT(IN) :: source_space_id ! Identifier of source
                                                ! dataspace
  INTEGER, INTENT(OUT) :: hdferr                ! Error code
                                                ! 0 on success and -1 on failure
END SUBROUTINE h5sextent_copy_f
```

Name: H5Sextent_equal

Signature:

htri_t `H5Sextent_equal`(*hid_t* `space1_id`, *hid_t* `space2_id`)

Purpose:

Determines whether two dataspace extents are equal.

Description:

`H5Sextent_equal` determines whether the dataspace extents of two dataspaces, `space1_id` and `space2_id`, are equal.

Parameters:

hid_t `space1_id`	First dataspace identifier.
hid_t `space2_id`	Second dataspace identifier.

Returns:

Returns `TRUE` if equal, `FALSE` if unequal, if successful; otherwise returns a negative value.

Fortran90 Interface: h5sextent_equal_f

```
SUBROUTINE h5sextent_equal_f(space1_id, space2_id, equal, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space1_id ! First dataspace identifier
  INTEGER(HID_T), INTENT(IN) :: space2_id ! Second dataspace identifier
  LOGICAL, INTENT(OUT) :: Equal           ! .TRUE. if equal, .FALSE. if unequal
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5sextent_equal_f
```

Name: H5Sget_select_bounds

Signature:

herr_t `H5Sget_select_bounds`(*hid_t* `space_id`, *hsize_t* *`start`, *hsize_t* *`end`)

Purpose:

Gets the bounding box containing the current selection.

Description:

`H5Sget_select_bounds` retrieves the coordinates of the bounding box containing the current selection and places them into user-supplied buffers.

The `start` and `end` buffers must be large enough to hold the dataspace rank number of coordinates.

The bounding box exactly contains the selection. I.e., if a 2-dimensional element selection is currently defined as containing the points (4,5), (6,8), and (10,7), then the bounding box will be (4, 5), (10, 8).

The bounding box calculation includes the current offset of the selection within the dataspace extent.

Calling this function on a `none` selection will return `FAIL`.

Parameters:

hid_t `space_id`	IN: Identifier of dataspace to query.
hsize_t *`start`	OUT: Starting coordinates of the bounding box.
hsize_t *`end`	OUT: Ending coordinates of the bounding box, i.e., the coordinates of the diagonally opposite corner.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

```
SUBROUTINE  h5sget_select_bounds_f(space_id, start, end, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id
                                   ! Dataspace identifier
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(OUT) :: start
                                   ! Starting coordinates of the bounding box
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(OUT) :: end
                                   ! Ending coordinates of the bounding box,
                                   ! i.e., the coordinates of the diagonally
                                   ! opposite corner
  INTEGER, INTENT(OUT) :: hdferr   ! Error code
END SUBROUTINE h5sget_select_bounds_f
```

History:

Release	**C**
1.6.0	The `start` and `end` parameters have changed from type *hsize_t* * to *hssize_t* *.

Name: H5Sget_select_elem_npoints

Signature:

hssize_t `H5Sget_select_elem_npoints`(*hid_t* `space_id`)

Purpose:

Gets the number of element points in the current selection.

Description:

`H5Sget_select_elem_npoints` returns the number of element points in the current dataspace selection.

Parameters:

hid_t `space_id`	IN: Identifier of dataspace to query.

Returns:

Returns the number of element points in the current dataspace selection if successful. Otherwise returns a negative value.

Fortran90 Interface: h5sget_select_elem_npoints_f

```
SUBROUTINE h5sget_select_elem_npoints_f(space_id, num_points, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id ! Dataspace identifier
  INTEGER, INTENT(OUT) :: num_points     ! Number of points in
                                         ! the current elements selection
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5sget_select_elem_npoints_f
```

Name: H5Sget_select_elem_pointlist

Signature:

herr_t `H5Sget_select_elem_pointlist`(*hid_t* `space_id`, *hsize_t* `startpoint`, *hsize_t* `numpoints`, *hsize_t* `*buf`)

Purpose:

Gets the list of element points currently selected.

Description:

`H5Sget_select_elem_pointlist` returns the list of element points in the current dataspace selection. Starting with the `startpoint`-th point in the list of points, `numpoints` points are put into the user's buffer. If the user's buffer fills up before `numpoints` points are inserted, the buffer will contain only as many points as fit.

The element point coordinates have the same dimensionality (rank) as the dataspace they are located within. The list of element points is formatted as follows:
<coordinate>, followed by
the next coordinate,
etc.
until all of the selected element points have been listed.

The points are returned in the order they will be iterated through when the selection is read/written from/to disk.

Parameters:

hid_t `space_id`	IN: Dataspace identifier of selection to query.
hsize_t `startpoint`	IN: Element point to start with.
hsize_t `numpoints`	IN: Number of element points to get.
hsize_t `*buf`	OUT: List of element points selected.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sget_select_elem_pointlist_f

```
SUBROUTINE h5sget_select_elem_pointlist_f(space_id, startpoint, num_points,
                                          buf, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: space_id   ! Dataspace identifier
  INTEGER(HSIZE_T), INTENT(IN) :: startpoint ! Element point to start with
  INTEGER, INTENT(OUT) :: num_points         ! Number of points to get in
                                             ! the current element selection
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(OUT) :: buf
                                             ! List of points selected
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
END SUBROUTINE h5sget_select_elem_pointlist_f
```

Name: H5Sget_select_hyper_blocklist

Signature:

herr_t `H5Sget_select_hyper_blocklist`(*hid_t* `space_id`, *hsize_t* `startblock`, *hsize_t* `numblocks`, *hsize_t* *`buf`)

Purpose:

Gets the list of hyperslab blocks currently selected.

Description:

`H5Sget_select_hyper_blocklist` returns a list of the hyperslab blocks currently selected. Starting with the `startblock`-th block in the list of blocks, `numblocks` blocks are put into the user's buffer. If the user's buffer fills up before `numblocks` blocks are inserted, the buffer will contain only as many blocks as fit.

The block coordinates have the same dimensionality (rank) as the dataspace they are located within. The list of blocks is formatted as follows:

<"start" coordinate>, immediately followed by
<"opposite" corner coordinate>, followed by
the next "start" and "opposite" coordinates,
etc.

until all of the selected blocks have been listed.

No guarantee is implied as the order in which blocks are listed.

Parameters:

hid_t `space_id`	IN: Dataspace identifier of selection to query.
hsize_t `startblock`	IN: Hyperslab block to start with.
hsize_t `numblocks`	IN: Number of hyperslab blocks to get.
hsize_t *`buf`	OUT: List of hyperslab blocks selected.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sget_select_hyper_blocklist_f

```
SUBROUTINE h5sget_select_hyper_blocklist_f(space_id, startblock, num_blocks,
                                           buf, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN)   :: space_id   ! Dataspace identifier
  INTEGER(HSIZE_T), INTENT(IN) :: startblock ! Hyperslab block to start with
                                             ! NOTE: numbering starts at 0
  INTEGER, INTENT(OUT) :: num_blocks         ! Number of hyperslab blocks to
                                             ! get in the current hyperslab
                                             ! selection
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(OUT) :: buf
                                             ! List of hyperslab blocks selected
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
END SUBROUTINE h5sget_select_hyper_blocklist_f
```

Name: H5Sget_select_hyper_nblocks

Signature:

hssize_t `H5Sget_select_hyper_nblocks`(*hid_t* `space_id`)

Purpose:

Get number of hyperslab blocks.

Description:

`H5Sget_select_hyper_nblocks` returns the number of hyperslab blocks in the current dataspace selection.

Parameters:

hid_t `space_id`	IN: Identifier of dataspace to query.

Returns:

Returns the number of hyperslab blocks in the current dataspace selection if successful. Otherwise returns a negative value.

Fortran90 Interface: h5sget_select_hyper_nblocks_f

```
SUBROUTINE h5sget_select_hyper_nblocks_f(space_id, num_blocks, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id ! Dataspace identifier
  INTEGER, INTENT(OUT) :: num_blocks     ! Number of hyperslab blocks in
                                         ! the current hyperslab selection
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5sget_select_hyper_nblocks_f
```

Name: H5Sget_select_npoints

Signature:

hssize_t `H5Sget_select_npoints`(*hid_t* `space_id`)

Purpose:

Determines the number of elements in a dataspace selection.

Description:

`H5Sget_select_npoints` determines the number of elements in the current selection of a dataspace.

Parameters:

hid_t `space_id`	Dataspace identifier.

Returns:

Returns the number of elements in the selection if successful; otherwise returns a negative value.

Fortran90 Interface: h5sget_select_npoints_f

```
SUBROUTINE h5sget_select_npoints_f(space_id, npoints, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id     ! Dataspace identifier
  INTEGER(HSSIZE_T), INTENT(OUT) :: npoints  ! Number of elements in the
                                             ! selection
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5sget_select_npoints_f
```

Name: H5Sget_select_type

Signature:

H5S_sel_type `H5Sget_select_type`(*hid_t* `space_id`)

Purpose:

Determines the type of the dataspace selection.

Description:

`H5Sget_select_type` retrieves the type of selection currently defined for the dataspace `space_id`.

Parameters:

hid_t `space_id`	Dataspace identifier.

Returns:

Returns the dataspace selection type, a value of the enumerated datatype `H5S_sel_type`, if successful. Valid return values are as follows:

`H5S_SEL_NONE`	No selection is defined.
`H5S_SEL_POINTS`	A sequence of points is selected.
`H5S_SEL_HYPERSLABS`	A hyperslab or compound hyperslab is selected.
`H5S_SEL_ALL`	The entire dataset is selected.

Otherwise returns a negative value.

Fortran90 Interface: h5sget_select_type_f

```
SUBROUTINE h5sget_select_type_f(space_id, type, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id ! Dataspace identifier
  INTEGER, INTENT(OUT) :: type           ! Selection type
                                         ! Valid values are:
                                         !    H5S_SEL_ERROR_F
                                         !    H5S_SEL_NONE_F
                                         !    H5S_SEL_POINTS_F
                                         !    H5S_SEL_HYPERSLABS_F
                                         !    H5S_SEL_ALL_F
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5sget_select_type_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Sget_simple_extent_dims

Signature:

int `H5Sget_simple_extent_dims`(*hid_t* `space_id`, *hsize_t* *`dims`, *hsize_t* *`maxdims`)

Purpose:

Retrieves dataspace dimension size and maximum size.

Description:

`H5Sget_simple_extent_dims` returns the size and maximum sizes of each dimension of a dataspace through the `dims` and `maxdims` parameters.

Either or both of `dims` and `maxdims` may be NULL.

If a value in the returned array `maxdims` is `H5S_UNLIMITED` (-1), the maximum size of that dimension is unlimited.

Parameters:

hid_t `space_id`	IN: Identifier of the dataspace object to query
hsize_t *`dims`	OUT: Pointer to array to store the size of each dimension.
hsize_t *`maxdims`	OUT: Pointer to array to store the maximum size of each dimension.

Returns:

Returns the number of dimensions in the dataspace if successful; otherwise returns a negative value.

Fortran90 Interface: h5sget_simple_extent_dims_f

```
SUBROUTINE h5sget_simple_extent_dims_f(space_id, dims, maxdims, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id   ! Dataspace identifier
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(OUT) :: dims
                                           ! Array to store dimension sizes
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(OUT) :: maxdims
                                           ! Array to store max dimension sizes
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! Dataspace rank on success
                                           ! and -1 on failure
END SUBROUTINE h5sget_simple_extent_dims_f
```

Name: H5Sget_simple_extent_ndims
Signature:
int `H5Sget_simple_extent_ndims`(*hid_t* `space_id`)
Purpose:
Determines the dimensionality of a dataspace.
Description:
`H5Sget_simple_extent_ndims` determines the dimensionality (or rank) of a dataspace.
Parameters:

hid_t `space_id`	Identifier of the dataspace

Returns:
Returns the number of dimensions in the dataspace if successful; otherwise returns a negative value.
Fortran90 Interface: h5sget_simple_extent_ndims_f

```
SUBROUTINE h5sget_simple_extent_ndims_f(space_id, rank, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id   ! Dataspace identifier
  INTEGER, INTENT(OUT) :: rank             ! Number of dimensions
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5sget_simple_extent_ndims_f
```

Name: H5Sget_simple_extent_npoints

Signature:

hssize_t `H5Sget_simple_extent_npoints`(*hid_t* `space_id`)

Purpose:

Determines the number of elements in a dataspace.

Description:

`H5Sget_simple_extent_npoints` determines the number of elements in a dataspace. For example, a simple 3-dimensional dataspace with dimensions 2, 3, and 4 would have 24 elements.

Parameters:

hid_t `space_id`	Identifier of the dataspace object to query

Returns:

Returns the number of elements in the dataspace if successful; otherwise returns 0.

Fortran90 Interface: h5sget_simple_extent_npoints_f

```
SUBROUTINE h5sget_simple_extent_npoints_f(space_id, npoints, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id    ! Dataspace identifier
  INTEGER(HSIZE_T), INTENT(OUT) :: npoints  ! Number of elements in dataspace
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5sget_simple_extent_npoints_f
```

Name: H5Sget_simple_extent_type

Signature:

H5S_class_t `H5Sget_simple_extent_type`(*hid_t* `space_id`)

Purpose:

Determine the current class of a dataspace.

Description:

`H5Sget_simple_extent_type` queries a dataspace to determine the current class of a dataspace.

The function returns a class name, one of the following: `H5S_SCALAR`, `H5S_SIMPLE`, or `H5S_NONE`.

Parameters:

hid_t `space_id`	Dataspace identifier.

Returns:

Returns a dataspace class name if successful; otherwise H5S_NO_CLASS (-1).

Fortran90 Interface: h5sget_simple_extent_type_f

```
SUBROUTINE h5sget_simple_extent_type_f(space_id, classtype, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id ! Dataspace identifier
  INTEGER, INTENT(OUT) :: classtype      ! Class type
                                         ! Possible values are:
                                         !    H5S_NO_CLASS_F
                                         !    H5S_SCALAR_F
                                         !    H5S_SIMPLE_F
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5sget_simple_extent_type_f
```

Name: H5Sis_simple

Signature:

htri_t `H5Sis_simple`(*hid_t* `space_id`)

Purpose:

Determines whether a dataspace is a simple dataspace.

Description:

`H5Sis_simple` determines whether a dataspace is a simple dataspace. [Currently, all dataspace objects are simple dataspaces, complex dataspace support will be added in the future]

Parameters:

hid_t `space_id`	Identifier of the dataspace to query

Returns:

When successful, returns a positive value, for `TRUE`, or `0` (zero), for `FALSE`. Otherwise returns a negative value.

Fortran90 Interface: h5sis_simple_f

```
SUBROUTINE h5sis_simple_f(space_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id    ! Dataspace identifier
  LOGICAL, INTENT(OUT) :: flag              ! Flag, indicates if dataspace
                                            ! is simple or not:
                                            ! TRUE or FALSE
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5sis_simple_f
```

Name: H5Soffset_simple

Signature:

herr_t `H5Soffset_simple`(*hid_t* `space_id`, *const hssize_t* *`offset`)

Purpose:

Sets the offset of a simple dataspace.

Description:

`H5Soffset_simple` sets the offset of a simple dataspace `space_id`. The `offset` array must be the same number of elements as the number of dimensions for the dataspace. If the `offset` array is set to `NULL`, the offset for the dataspace is reset to 0.

This function allows the same shaped selection to be moved to different locations within a dataspace without requiring it to be redefined.

Parameters:

hid_t `space_id`	IN: The identifier for the dataspace object to reset.
const hssize_t *`offset`	IN: The offset at which to position the selection.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5soffset_simple_f

```
SUBROUTINE h5soffset_simple_f(space_id, offset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id    ! Dataspace identifier
  INTEGER(HSSIZE_T), DIMENSION(*), INTENT(IN) ::  offset
                                            ! The offset at which to position
                                            ! the selection
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                            ! 0 on success and -1 on failure
END SUBROUTINE h5soffset_simple_f
```

Last modified: 6 April 2009

Name: H5Sselect_all

Signature:

herr_t `H5Sselect_all`(*hid_t* `dspace_id`)

Purpose:

Selects an entire dataspace.

Description:

`H5Sselect_all` selects the entire extent of the dataspace `dspace_id`.

More specifically, `H5Sselect_all` sets the selection type to `H5S_SEL_ALL`, which specifies the entire dataspace anywhere it is applied.

Parameters:

hid_t `dspace_id`	IN: The identifier for the dataspace for which the selection is being made.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

See Also:

`H5Sget_select_type`

Fortran90 Interface: h5sselect_all_f

```
SUBROUTINE h5sselect_all_f(dspace_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dspace_id ! Dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5sselect_all_f
```

Name: H5Sselect_elements

Signature:

herr_t `H5Sselect_elements`(*hid_t* `space_id`, *H5S_seloper_t* `op`, *size_t* `num_elements`, *const hsize_t* *`coord`)

Purpose:

Selects array elements to be included in the selection for a dataspace.

Description:

`H5Sselect_elements` selects array elements to be included in the selection for the `space_id` dataspace.

The number of elements selected is set in the `num_elements` parameter.

The `coord` array is a two-dimensional array of size `num_elements` by *`dataspace_rank`* containing a list of of zero-based values specifying the coordinates in the dataset of the selected elements. The order of the element coordinates in the `coord` array specifies the order in which the array elements are iterated through when I/O is performed. Duplicate coordinate locations are not checked for.

The selection operator `op` determines how the new selection is to be combined with the previously existing selection for the dataspace. The following operators are supported:

`H5S_SELECT_SET`	Replaces the existing selection with the parameters from this call. Overlapping blocks are not supported with this operator. Adds the new selection to the existing selection.
`H5S_SELECT_APPEND`	Adds the new selection following the last element of the existing selection.
`H5S_SELECT_PREPEND`	Adds the new selection preceding the first element of the existing selection.

Parameters:

hid_t `space_id`	Identifier of the dataspace.
H5S_seloper_t `op`	Operator specifying how the new selection is to be combined with the existing selection for the dataspace.
size_t `num_elements`	Number of elements to be selected.
const hsize_t *`coord`	A 2-dimensional array of 0-based values specifying the coordinates of the elements being selected.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sselect_elements_f

```
SUBROUTINE h5sselect_elements_f(space_id, operator, rank, num_elements,
                                coord, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id  ! Dataspace identifier
  INTEGER, INTENT(IN) :: operator         ! Flag, valid values are:
                                          !    H5S_SELECT_SET_F
                                          !    H5S_SELECT_APPEND_F
                                          !    H5S_SELECT_PREPEND_F
  INTEGER, INTENT(IN) :: rank             ! Number of dataspace
                                          ! dimensions
  INTEGER(SIZE_T), INTENT(IN) :: num_elements
                                          ! Number of elements to be
                                          ! selected
```

```
    INTEGER(HSIZE_T), DIMENSION(rank,num_elements), INTENT(IN) :: coord
                                            ! A 1-based array containing the
                                            ! coordinates of the selected
                                            ! elements
                                            ! NOTE: Reversed dimension declaration
                                            ! compared to the C specification
                                            ! of coord(num_elements, rank)
    INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                            ! 0 on success and -1 on failure
  END SUBROUTINE h5sselect_elements_f
```

History:

Release	**C**	**Fortran90**
1.6.4	`coord` parameter type changed to *const hsize_t*.	`coord` parameter type changed to `INTEGER(HSIZE_T)`.

Name: H5Sselect_hyperslab

Signature:

herr_t `H5Sselect_hyperslab`(*hid_t* `space_id`, *H5S_seloper_t* `op`, *const hsize_t* *`start`, *const hsize_t* *`stride`, *const hsize_t* *`count`, *const hsize_t* *`block`)

Purpose:

Selects a hyperslab region to add to the current selected region.

Description:

`H5Sselect_hyperslab` selects a hyperslab region to add to the current selected region for the dataspace specified by `space_id`.

The `start`, `stride`, `count`, and `block` arrays must be the same size as the rank of the dataspace. For example, if the dataspace is 4-dimensional, each of these parameters must be a 1-dimensional array of size `4`.

The selection operator `op` determines how the new selection is to be combined with the already existing selection for the dataspace. The following operators are supported:

`H5S_SELECT_SET`	Replaces the existing selection with the parameters from this call. Overlapping blocks are not supported with this operator.
`H5S_SELECT_OR`	Adds the new selection to the existing selection. (Binary OR)
`H5S_SELECT_AND`	Retains only the overlapping portions of the new selection and the existing selection. (Binary AND)
`H5S_SELECT_XOR`	Retains only the elements that are members of the new selection or the existing selection, excluding elements that are members of both selections. (Binary exclusive-OR, XOR)
`H5S_SELECT_NOTB`	Retains only elements of the existing selection that are not in the new selection.
`H5S_SELECT_NOTA`	Retains only elements of the new selection that are not in the existing selection.

The `start` array specifies the offset of the starting element of the specified hyperslab.

The `stride` array chooses array locations from the dataspace with each value in the `stride` array determining how many elements to move in each dimension. Setting a value in the `stride` array to `1` moves to each element in that dimension of the dataspace; setting a value of `2` in allocation in the `stride` array moves to every other element in that dimension of the dataspace. In other words, the `stride` determines the number of elements to move from the `start` location in each dimension. Stride values of `0` are not allowed. If the `stride` parameter is `NULL`, a contiguous hyperslab is selected (as if each value in the `stride` array were set to `1`).

The `count` array determines how many blocks to select from the dataspace, in each dimension.

The `block` array determines the size of the element block selected from the dataspace. If the `block` parameter is set to `NULL`, the block size defaults to a single element in each dimension (as if each value in the `block` array were set to `1`).

For example, consider a 2-dimensional dataspace with hyperslab selection settings as follows: the `start` offset is specified as [1,1], `stride` is [4,4], `count` is [3,7], and `block` is [2,2]. In C, these settings will specify a hyperslab consisting of 21 2x2 blocks of array elements starting with location (1,1) with the selected blocks at locations (1,1), (5,1), (9,1), (1,5), (5,5), etc.; in Fortran, they will specify a hyperslab consisting of 21 2x2 blocks of array elements starting with location (2,2) with the selected blocks at locations (2,2), (6,2), (10,2), (2,6), (6,6), etc.

Regions selected with this function call default to C order iteration when I/O is performed.

Parameters:

hid_t `space_id`	IN: Identifier of dataspace selection to modify
H5S_seloper_t `op`	IN: Operation to perform on current selection.
const hsize_t *`start`	IN: Offset of start of hyperslab
const hsize_t *`count`	IN: Number of blocks included in hyperslab.
const hsize_t *`stride`	IN: Hyperslab stride.
const hsize_t *`block`	IN: Size of block in hyperslab.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sselect_hyperslab_f

```
SUBROUTINE h5sselect_hyperslab_f(space_id, operator, start, count,
                                 hdferr, stride, block)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id  ! Dataspace identifier
  INTEGER, INTENT(IN) :: op               ! Flag, valid values are:
                                          !    H5S_SELECT_SET_F
                                          !    H5S_SELECT_OR_F
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN) :: start
                                          ! Offset of start of hyperslab
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN) :: count
                                          ! Number of blocks to select
                                          ! from dataspace
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
  INTEGER(HSIZE_T), DIMENSION(*), OPTIONAL, INTENT(IN) :: stride
                                          ! Array of how many elements to
                                          ! move in each direction
  INTEGER(HSIZE_T), DIMENSION(*), OPTIONAL, INTENT(IN) :: block
                                          ! Size of the element block
END SUBROUTINE h5sselect_hyperslab_f
```

History:

Release	**C**	**Fortran90**
1.6.4	`start[]` parameter type changed to *const hsize_t*.	`start` parameter type changed to `INTEGER(HSIZE_T)`.

Name: H5Sselect_none

Signature:

herr_t `H5Sselect_none`(*hid_t* `space_id`)

Purpose:

Resets the selection region to include no elements.

Description:

`H5Sselect_none` resets the selection region for the dataspace `space_id` to include no elements.

Parameters:

hid_t `space_id`	IN: The identifier for the dataspace in which the selection is being reset.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sselect_none_f

```
SUBROUTINE h5sselect_none_f(space_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id  ! Dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5sselect_none_f
```

Name: H5Sselect_valid

Signature:

htri_t `H5Sselect_valid`(*hid_t* `space_id`)

Purpose:

Verifies that the selection is within the extent of the dataspace.

Description:

`H5Sselect_valid` verifies that the selection for the dataspace `space_id` is within the extent of the dataspace if the current offset for the dataspace is used.

Parameters:

hid_t `space_id`	The identifier for the dataspace being queried.

Returns:

Returns a positive value, for `TRUE`, if the selection is contained within the extent or `0` (zero), for `FALSE`, if it is not. Returns a negative value on error conditions such as the selection or extent not being defined.

Fortran90 Interface: h5sselect_valid_f

```
SUBROUTINE h5sselect_valid_f(space_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id  ! Dataspace identifier
  LOGICAL, INTENT(OUT) :: flag            ! TRUE if the selection is
                                          ! contained within the extent,
                                          ! FALSE otherwise.
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5sselect_valid_f
```

Name: H5Sset_extent_none

Signature:

herr_t `H5Sset_extent_none`(*hid_t* `space_id`)

Purpose:

Removes the extent from a dataspace.

Description:

`H5Sset_extent_none` removes the extent from a dataspace and sets the type to `H5S_NO_CLASS`.

Parameters:

hid_t `space_id`	The identifier for the dataspace from which the extent is to be removed.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sset_extent_none_f

```
SUBROUTINE h5sset_extent_none_f(space_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id  ! Dataspace identifier
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5sset_extent_none_f
```

Name: H5Sset_extent_simple

Signature:

herr_t `H5Sset_extent_simple`(*hid_t* `space_id`, *int* `rank`, *const hsize_t* *`current_size`, *const hsize_t* *`maximum_size`)

Purpose:

Sets or resets the size of an existing dataspace.

Description:

`H5Sset_extent_simple` sets or resets the size of an existing dataspace.

`rank` is the dimensionality, or number of dimensions, of the dataspace.

`current_size` is an array of size `rank` which contains the new size of each dimension in the dataspace. `maximum_size` is an array of size `rank` which contains the maximum size of each dimension in the dataspace.

Any previous extent is removed from the dataspace, the dataspace type is set to `H5S_SIMPLE`, and the extent is set as specified.

Parameters:

hid_t `space_id`	Dataspace identifier.
int `rank`	Rank, or dimensionality, of the dataspace.
const hsize_t *`current_size`	Array containing current size of dataspace.
const hsize_t *`maximum_size`	Array containing maximum size of dataspace.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5sset_extent_simple_f

```
SUBROUTINE h5sset_extent_simple_f(space_id, rank, current_size, 
                                  maximum_size, hdferr) 
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: space_id     ! Dataspace identifier 
  INTEGER, INTENT(IN) :: rank                ! Dataspace rank 
  INTEGER(HSIZE_T), DIMENSION(rank), INTENT(IN) :: current_size 
                                             ! Array with the new sizes
                                             ! of dimensions 
  INTEGER(HSIZE_T), DIMENSION(rank), INTENT(IN) ::  
                                             ! Array with the new maximum
                                             ! sizes of dimensions 
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5sset_extent_simple_f
```

H5T: Datatype Interface

Datatype Object API Functions

These functions create and manipulate the datatype which describes elements of a dataset.
In the following lists, italic type indicates a configurable macro.

The C Interfaces:

General Datatype Operations

- H5Tcreate
- *H5Topen*
- H5Topen1 *
- H5Topen2
- *H5Tcommit*
- H5Tcommit1 *
- H5Tcommit2
- H5Tcommit_anon
- H5Tcommitted
- H5Tcopy
- H5Tequal
- H5Tlock
- H5Tget_class
- H5Tget_create_plist
- H5Tget_size
- H5Tget_super
- H5Tget_native_type
- H5Tdetect_class
- H5Tclose

Conversion Functions

- H5Tconvert
- H5Tfind
- H5Tcompiler_conv
- H5Tregister
- H5Tunregister
- H5Tdecode
- H5Tencode

Atomic Datatype Properties

- H5Tset_size
- H5Tget_order
- H5Tset_order
- H5Tget_precision
- H5Tset_precision
- H5Tget_offset
- H5Tset_offset
- H5Tget_pad
- H5Tset_pad
- H5Tget_sign
- H5Tset_sign
- H5Tget_fields
- H5Tset_fields
- H5Tget_ebias
- H5Tset_ebias
- H5Tget_norm
- H5Tset_norm
- H5Tget_inpad
- H5Tset_inpad
- H5Tget_cset
- H5Tset_cset
- H5Tget_strpad
- H5Tset_strpad

Array Datatypes

- *H5Tarray_create*
- H5Tarray_create1 *
- H5Tarray_create2
- H5Tget_array_ndims
- *H5Tget_array_dims*
- H5Tget_array_dims1 *
- H5Tget_array_dims2

Compound Datatype Properties

- H5Tget_nmembers
- H5Tget_member_class
- H5Tget_member_name
- H5Tget_member_index
- H5Tget_member_offset
- H5Tget_member_type
- H5Tinsert
- H5Tpack

Variable-length Datatypes

- H5Tvlen_create
- H5Tis_variable_str

Opaque Datatypes

- H5Tset_tag
- H5Tget_tag

Enumeration Datatypes

- H5Tenum_create
- H5Tenum_insert
- H5Tenum_nameof
- H5Tenum_valueof
- H5Tget_member_value
- H5Tget_nmembers
- H5Tget_member_name
- H5Tget_member_index

* *Use of these functions is deprecated in Release 1.8.0.*

Alphabetical Listing

- *H5Tarray_create*
- H5Tarray_create1 *
- H5Tarray_create2
- H5Tclose
- *H5Tcommit*
- H5Tcommit1 *
- H5Tcommit2
- H5Tcommit_anon
- H5Tcommitted
- H5Tcompiler_conv
- H5Tconvert
- H5Tcopy
- H5Tcreate
- H5Tdecode
- H5Tdetect_class
- H5Tencode
- H5Tenum_create
- H5Tenum_insert
- H5Tenum_nameof
- H5Tenum_valueof
- H5Tequal
- H5Tfind
- *H5Tget_array_dims*
- H5Tget_array_dims1 *
- H5Tget_array_dims2
- H5Tget_array_ndims
- H5Tget_class
- H5Tget_create_plist
- H5Tget_cset
- H5Tget_ebias
- H5Tget_fields
- H5Tget_inpad
- H5Tget_member_class
- H5Tget_member_index
- H5Tget_member_name
- H5Tget_member_offset
- H5Tget_member_type
- H5Tget_member_value
- H5Tget_native_type
- H5Tget_nmembers
- H5Tget_norm
- H5Tget_offset
- H5Tget_order
- H5Tget_pad
- H5Tget_precision
- H5Tget_sign
- H5Tget_size
- H5Tget_strpad
- H5Tget_super
- H5Tget_tag
- H5Tinsert
- H5Tis_variable_str
- H5Tlock
- *H5Topen*
- H5Topen1 *
- H5Topen2
- H5Tpack
- H5Tregister
- H5Tset_cset
- H5Tset_ebias
- H5Tset_fields
- H5Tset_inpad
- H5Tset_norm
- H5Tset_offset
- H5Tset_order
- H5Tset_pad
- H5Tset_precision
- H5Tset_sign
- H5Tset_size
- H5Tset_strpad
- H5Tset_tag
- H5Tunregister
- H5Tvlen_create

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

General Datatype Operations
- h5tcreate_f
- h5tdecode_f
- h5tencode_f
- h5topen_f
- h5tcommit_f
- h5tcommit_anon_f
- h5tcommitted_f
- H5tcompiler_conv_f
- h5tcopy_f
- h5tequal_f
- h5tget_create_plist_f
- h5tget_class_f
- h5tget_size_f
- h5tget_super_f
- h5tclose_f

Enumeration Datatypes
- h5tenum_create_f
- h5tenum_insert_f
- h5tenum_nameof_f
- h5tenum_valueof_f
- h5tget_member_value_f
- h5tget_native_type_f
- h5tget_nmembers_f
- h5tget_member_name_f
- h5tget_member_index_f

Atomic Datatype Properties
- h5tset_size_f
- h5tget_order_f
- h5tset_order_f
- h5tget_precision_f
- h5tset_precision_f
- h5tget_offset_f
- h5tset_offset_f
- h5tget_pad_f
- h5tset_pad_f
- h5tget_sign_f
- h5tset_sign_f
- h5tget_fields_f
- h5tset_fields_f
- h5tget_ebiass_f
- h5tset_ebiass_f
- h5tget_norm_f
- h5tset_norm_f
- h5tget_inpad_f
- h5tset_inpad_f
- h5tget_cset_f
- h5tset_cset_f
- h5tget_strpad_f
- h5tset_strpad_f

Array Datatypes
- h5tarray_create_f
- h5tget_array_ndims_f
- h5tget_array_dims_f

Compound Datatype Properties
- h5tget_nmembers_f
- h5tget_member_class_f
- h5tget_member_name_f
- h5tget_member_index_f
- h5tget_member_offset_f
- h5tget_member_type_f
- h5tinsert_f
- h5tpack_f

Variable-length Datatypes
- h5tvlen_create_f
- h5tis_variable_str_f

Opaque Datatypes
- h5tset_tag_f
- h5tget_tag_f

The Datatype interface, H5T, provides a mechanism to describe the storage format of individual data points of a data set and is hopefully designed in such a way as to allow new features to be easily added without disrupting applications that use the data type interface. A dataset (the H5D interface) is composed of a collection or raw data points of homogeneous type organized according to the data space (the H5S interface).

A datatype is a collection of datatype properties, all of which can be stored on disk, and which when taken as a whole, provide complete information for data conversion to or from that datatype. The interface provides functions to set and query properties of a datatype.

A *data point* is an instance of a *datatype*, which is an instance of a *type class*. We have defined a set of type classes and properties which can be extended at a later time. The atomic type classes are those which describe types which cannot be decomposed at the datatype interface level; all other classes are compound.

See *The Datatype Interface (H5T)* in the *HDF5 User's Guide* for further information, including a complete list of all supported datatypes.

Name: H5Tarray_create

Signatures:

hid_t `H5Tarray_create(` *hid_t* `base_type_id`, *int* `rank`, *const hsize_t* `dims[/*rank*/]`, *const int* `perm[/*rank*/]`) [1]

hid_t `H5Tarray_create(` *hid_t* `base_typ_id`, *unsigned* `rank`, *const hsize_t* `dims[/*rank*/]`,) [2]

Purpose:

Creates an array datatype object.

Description:

`H5Tarray_create` is a macro that is mapped to either `H5Tarray_create1` or `H5Tarray_create2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

◊ The `H5Tarray_create` macro will be mapped to `H5Tarray_create1` and will use the `H5Tarray_create1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.

◊ The `H5Tarray_create` macro mapped to `H5Tarray_create2` and will use the `H5Tarray_create2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Tarray_create` is mapped to the most recent version of the function, currently `H5Tarray_create2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Tarray_create` will be mapped to `H5Tarray_create1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Tarray_create` mapping**
<u>Global settings</u>	
No compatibility flag	`H5Tarray_create2`
Enable deprecated symbols	`H5Tarray_create2`
Disable deprecated symbols	`H5Tarray_create2`
Emulate Release 1.6 interface	`H5Tarray_create1`
<u>Function-level macros</u>	
`H5Tarray_create_vers = 2`	`H5Tarray_create2`
`H5Tarray_create_vers = 1`	`H5Tarray_create1`

Interface history: Signature [1] above is the original `H5Tarray_create` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Tarray_create1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Tarray_create2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface: h5tarray_create_f

```
SUBROUTINE h5tarray_create_f(base_id, rank, dims, type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: base_id   ! Identifier of array base datatype
  INTEGER, INTENT(IN)        ::  rank     ! Rank of the array
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN) :: dims
                                          ! Sizes of each array dimension
  INTEGER(HID_T), INTENT(OUT) :: type_id  ! Identifier of the array datatype
  INTEGER, INTENT(OUT)        :: hdferr   ! Error code
END SUBROUTINE h5tarray_create_f
```

History:

Release	**C**
1.8.0	The function `H5Tarray_create` renamed to `H5Tarray_create1` and deprecated in this release. The macro `H5Tarray_create` and the function `H5Tarray_create2` introduced in this release.

Name: H5Tarray_create1

Signature:

hid_t `H5Tarray_create1(` *hid_t* `base_type_id`, *int* `rank`, *const hsize_t* `dims[/*rank*/]`, *const int* `perm[/*rank*/]`)

Purpose:

Creates an array datatype object.

Notice:

This function is renamed from `H5Tarray_create` and deprecated in favor of the function `H5Tarray_create2` or the new macro `H5Tarray_create`.

Description:

`H5Tarray_create1` creates a new array datatype object.

`base_type_id` is the datatype of every element of the array, i.e., of the number at each position in the array.

`rank` is the number of dimensions and the size of each dimension is specified in the array `dims`. The value of `rank` is currently limited to `H5S_MAX_RANK` and must be greater than `0` (zero). All dimension sizes specified in `dims` must be greater than `0` (zero).

The array `perm` is designed to contain the dimension permutation, i.e. C versus FORTRAN array order. ***(The parameter `perm` is currently unused and is not yet implemented.)***

Parameters:

hid_t `base_type_id`	IN: Datatype identifier for the array base datatype.
int `rank`	IN: Rank of the array.
const hsize_t `dims[/*rank*/]`	IN: Size of each array dimension.
const int `perm[/*rank*/]`	IN: Dimension permutation. ***(Currently not implemented.)***

Returns:

Returns a valid datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5tarray_create_f

```
SUBROUTINE h5tarray_create_f(base_id, rank, dims, type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: base_id   ! Identifier of array base datatype
  INTEGER, INTENT(IN)        ::  rank     ! Rank of the array
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN) :: dims
                                          ! Sizes of each array dimension
  INTEGER(HID_T), INTENT(OUT) :: type_id  ! Identifier of the array datatype
  INTEGER, INTENT(OUT)        :: hdferr   ! Error code
END SUBROUTINE h5tarray_create_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.
1.8.0	Function `H5Tarray_create` renamed to `H5Tarray_create1` and deprecated in this release.

Last modified: 9 April 2009

Name: H5Tarray_create2

Signature:

hid_t `H5Tarray_create2`(*hid_t* `base_type_id`, *unsigned* `rank`, *const hsize_t* `dims[/*rank*/]`,)

Purpose:

Creates an array datatype object.

Description:

`H5Tarray_create2` creates a new array datatype object.

`base_type_id` is the datatype of every element of the array, i.e., of the number at each position in the array.

`rank` is the number of dimensions and the size of each dimension is specified in the array `dims`. The value of `rank` is currently limited to `H5S_MAX_RANK` and must be greater than `0` (zero). All dimension sizes specified in `dims` must be greater than `0` (zero).

Parameters:

hid_t `base_type_id`	IN: Datatype identifier for the array base datatype.
unsigned `rank`	IN: Rank of the array.
const hsize_t `dims[/*rank*/]`	IN: Size of each array dimension.

Returns:

Returns a valid datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**Change**
1.8.0	C function introduced in this release.

Name: H5Tclose

Signature:

herr_t `H5Tclose`(*hid_t* `dtype_id`)

Purpose:

Releases a datatype.

Description:

`H5Tclose` releases a datatype. Further access through the datatype identifier is illegal. Failure to release a datatype with this call will result in resource leaks.

Parameters:

hid_t `dtype_id`	Identifier of datatype to release.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tclose_f

```
SUBROUTINE h5tclose_f(type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5tclose_f
```

Name: H5Tcommit

Signature:

herr_t `H5Tcommit`(*hid_t* `loc_id`, *const char ** `name`, *hid_t* `dtype_id`)

hid_t `H5Tcommit`(*hid_t* `loc_id`, *const char ** `name`, *hid_t* `dtype_id`, *hid_t* `lcpl_id`, *hid_t* `tcpl_id`, *hid_t* `tapl_id`)

Purpose:

Commits a transient datatype, linking it into the file and creating a new named datatype.

Description:

`H5Tcommit` is a macro that is mapped to either `H5Tcommit1` or `H5Tcommit2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Tcommit` is mapped to the most recent version of the function, currently `H5Tcommit2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Tcommit` will be mapped to `H5Tcommit1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Tcommit` mapping**
Global settings	
No compatibility flag	`H5Tcommit2`
Enable deprecated symbols	`H5Tcommit2`
Disable deprecated symbols	`H5Tcommit2`
Emulate Release 1.6 interface	`H5Tcommit1`
Function-level macros	
`H5Tcommit_vers = 2`	`H5Tcommit2`
`H5Tcommit_vers = 1`	`H5Tcommit1`

Fortran90 Interface: h5tcommit_f

```
SUBROUTINE h5tcommit_f(loc_id, name, type_id, hdferr, &
                       lcpl_id, tcpl_id, tapl_id )
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id  ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Datatype name within file or group
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: lcpl_id
                                        ! Link creation property list
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: tcpl_id
                                        ! Datatype creation property list
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: tapl_id
                                        ! Datatype access property list
END SUBROUTINE h5tcommit_f
```

History:

Release	**C**
1.8.0	The function `H5Tcommit` renamed to `H5Tcommit1` and deprecated in this release. The macro `H5Tcommit` and the function `H5Tcommit2` introduced in this release.

Last modified: 18 May 2009

Name: H5Tcommit1

Signature:

herr_t `H5Tcommit1`(*hid_t* `loc_id`, *const char* * `name`, *hid_t* `dtype_id`)

Purpose:

Commits a transient datatype to a file, creating a new named datatype.

Notice:

This function is deprecated in favor of the function `H5Tcommit2`.

Description:

`H5Tcommit1` commits the transient datatype (not immutable) to a file, turning it into a named datatype.

The datatype `dtype_id` is committed as a named datatype at the location `loc_id`, which is either a file or group identifier, with the name `name`.

`name` can be a relative path based at `loc_id` or an absolute path from the root of the file. If any of the groups specified in that path do not already exist, the dataset must be created with `H5Tcommit_anon` and linked into the file structure with `H5Lcreate_hard`.

As is the case for any object in a group, the length of the name of a named datatype is not limited.

See `H5Tcommit_anon` for a discussion of the differences between `H5Tcommit` and `H5Tcommit_anon`.

Parameters:

hid_t `loc_id`	IN: A file or group identifier.
const char * `name`	IN: A datatype name.
hid_t `dtype_id`	IN: A datatype identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Tcommit`.

History:

Release	**C**
1.8.0	The function `H5Tcommit` renamed to `H5Tcommit1` and deprecated in this release.

Name: H5Tcommit2

Signature:

herr_t `H5Tcommit2`(*hid_t* `loc_id`, *const char* *`name`, *hid_t* `dtype_id`, *hid_t* `lcpl_id`, *hid_t* `tcpl_id`, *hid_t* `tapl_id`)

Purpose:

Commits a transient datatype, linking it into the file and creating a new named datatype.

Description:

`H5Tcommit2` saves a transient datatype as an immutable named datatype in a file. The datatype specified by `dtype_id` is committed to the file with the name `name` at the location specified by `loc_id` and with the datatype creation and access property lists `tcpl_id` and `tapl_id`, respectively.

`loc_id` may be a file identifier, or a group identifier within that file. `name` may be either an absolute path in the file or a relative path from `loc_id` naming the newly-commited datatype.

The link creation property list, `lcpl_id`, governs creation of the link(s) by which the new named datatype is accessed and the creation of any intermediate groups that may be missing.

Once commited, this datatype may be used to define the datatype of any other dataset or attribute in the file.

Parameters:

hid_t `loc_id`	IN: Location identifier
const char *`name`	IN: Name given to committed datatype
hid_t `dtype_id`	IN: Identifier of datatype to be committed
hid_t `lcpl_id`	IN: Link creation property list
hid_t `tcpl_id`	IN: Datatype creation property list
hid_t `tapl_id`	IN: Datatype access property list

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Tcommit`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Last modified: 18 May 2009

Name: H5Tcommit_anon

Signature:

herr_t `H5Tcommit_anon`(*hid_t* `loc_id`, *hid_t* `dtype_id`, *hid_t* `tcpl_id`, *hid_t* `tapl_id`)

Purpose:

Commits a transient datatype to a file, creating a new named datatype, but does not link it into the file structure.

Description:

`H5Tcommit_anon` commits a transient datatype (not immutable) to a file, turning it into a named datatype with the specified creation and property lists. With default property lists, `H5P_DEFAULT`, `H5Tcommit_anon` provides similar functionality to that of `H5Tcommit`, with the differences described below.

The datatype access property list identifier, `tapl_id`, is provided for future functionality and is not used at this time. This parameter should always be passed as the value `H5P_DEFAULT`.

Note that `H5Tcommit_anon` does not link this newly-committed datatype into the file. After the `H5Tcommit_anon` call, the datatype identifier `dtype_id` *must* be linked into the HDF5 file structure with `H5Lcreate_hard` or it will be deleted from the file when the file is closed.

The differences between this function and `H5Tcommit` are as follows:

◊ `H5Tcommit_anon` explicitly includes property lists, which provides for greater control of the creation process and of the properties of the new named datatype. `H5Tcommit` always uses default properties.

◊ `H5Tcommit_anon` neither provides the new named datatype's name nor links it into the HDF5 file structure; those actions must be performed separately through a call to `H5Lcreate_hard`, which offers greater control over linking.

Parameters:

hid_t `loc_id`	IN: A file or group identifier specifying the file in which the new named datatype is to be created.
hid_t `dtype_id`	IN: A datatype identifier.
hid_t `tcpl_id`	IN: A datatype creation property list identifier. (`H5P_DEFAULT` for the default property list.)
hid_t `tapl_id`	IN: A datatype access property list identifier. *Currently unused; should always be passed as the value* `H5P_DEFAULT`.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tcommit_anon_f

```
SUBROUTINE h5tcommit_anon_f(loc_id, dtype_id, hdferr, tcpl_id, tapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id ! A file or group identifier specifying
                                       ! the file in which the new named
                                       ! datatype is to be created.
  INTEGER(HID_T), INTENT(IN) :: dtype_id
                                       ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: tcpl_id
```

```
                                       ! A datatype creation property
                                       ! list identifier.
                                       !  H5P_DEFAULT_F = default property list
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: tapl_id
                                       ! A datatype access property list id
END SUBROUTINE h5tcommit_anon_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Tcommitted

Signature:

htri_t`H5Tcommitted`(*hid_t* `dtype_id`)

Purpose:

Determines whether a datatype is a named type or a transient type.

Description:

`H5Tcommitted` queries a type to determine whether the type specified by the `dtype_id` identifier is a named type or a transient type. If this function returns a positive value, then the type is named (that is, it has been committed, perhaps by some other application). Datasets which return committed datatypes with `H5Dget_type()` are able to share the datatype with other datasets in the same file.

Parameters:

hid_t `dtype_id`	IN: Datatype identifier.

Returns:

When successful, returns a positive value, for `TRUE`, if the datatype has been committed, or `0` (zero), for `FALSE`, if the datatype has not been committed. Otherwise returns a negative value.

Fortran90 Interface: h5tcommitted_f

```
SUBROUTINE h5tcommitted_f(dtype_id, committed, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dtype_id
                                    ! A datatype identifier
  LOGICAL, INTENT(OUT) :: committed ! .TRUE., if the datatype committed
                                    ! .FALSE., if the datatype not committed.
  INTEGER, INTENT(OUT) :: hdferr    ! Error code
                                    ! 0 on success and -1 on failure
END SUBROUTINE h5tcommitted_f
```

Name: H5Tcompiler_conv
Signature:

htri_t `H5Tcompiler_conv`(*hid_t* `src_id`, *hid_t* `dst_id`)

Purpose:

Check whether the library s default conversion is hard conversion.

Description:

`H5Tcompiler_conv` finds out whether the library s conversion function from type `src_id` to type `dst_id` is a compiler (hard) conversion. A compiler conversion uses compiler s casting; a library (soft) conversion uses the library s own conversion function.

Parameters:

hid_t `src_id`	IN: Identifier for the source datatype.
hid_t `dst_id`	IN: Identifier for the destination datatype.

Returns:

Returns TRUE for compiler conversion, FALSE for library conversion, FAIL for the function s failure.

Fortran90 Interface: h5tcompiler_conv_f

```
SUBROUTINE h5tcompiler_conv_f( src_id, dst_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: src_id ! Id for the source datatype.
  INTEGER(HID_T), INTENT(IN) :: dst_id ! Id for the destination datatype.
  LOGICAL, INTENT(OUT) :: flag         !   .TRUE. for compiler conversion,
                                       !   .FALSE. for library conversion
  INTEGER, INTENT(OUT) :: hdferr       ! Error code:
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5tcompiler_conv_f
```

Name: H5Tconvert

Signature:

herr_t `H5Tconvert`(*hid_t* `src_id`, *hid_t* `dst_id`, *size_t* `nelmts`, *void* *`buf`, *void* *`background`, *hid_t* `plist_id`)

Purpose:

Converts data from between specified datatypes.

Description:

`H5Tconvert` converts `nelmts` elements from the type specified by the `src_id` identifier to type `dst_id`. The source elements are packed in `buf` and on return the destination will be packed in `buf`. That is, the conversion is performed in place. The optional background buffer is an array of `nelmts` values of destination type which are merged with the converted values to fill in cracks (for instance, `background` might be an array of structs with the `a` and `b` fields already initialized and the conversion of `buf` supplies the `c` and `d` field values).

The parameter `plist_id` contains the dataset transfer property list identifier which is passed to the conversion functions. As of Release 1.2, this parameter is only used to pass along the variable-length datatype custom allocation information.

Parameters:

hid_t `src_id`	Identifier for the source datatype.
hid_t `dst_id`	Identifier for the destination datatype.
size_t `nelmts`	Size of array `buf`.
void *`buf`	Array containing pre- and post-conversion values.
void *`background`	Optional background buffer.
hid_t `plist_id`	Dataset transfer property list identifier.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.3	`nelmts` parameter type changed to *size_t*.
1.4.0	`nelmts` parameter type changed to *hsize_t*.

Name: H5Tcopy
Signature:

hid_t `H5Tcopy`(*hid_t* `dtype_id`)

Purpose:

Copies an existing datatype.

Description:

`H5Tcopy` copies an existing datatype. The returned type is always transient and unlocked.

The `dtype_id` argument can be either a datatype identifier, a predefined datatype (defined in `H5Tpublic.h`), or a dataset identifier. If `dtype_id` is a dataset identifier instead of a datatype identifier, then this function returns a transient, modifiable datatype which is a copy of the dataset's datatype.

The datatype identifier returned should be released with `H5Tclose` or resource leaks will occur.

Parameters:

hid_t `dtype_id`	Identifier of datatype to copy. Can be a datatype identifier, a predefined datatype (defined in `H5Tpublic.h`), or a dataset identifier.

Returns:

Returns a datatype identifier if successful; otherwise returns a negative value

Fortran90 Interface: h5tcopy_f

```
SUBROUTINE h5tcopy_f(type_id, new_type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id      ! Datatype identifier
  INTEGER(HID_T), INTENT(OUT) :: new_type_id ! Identifier of datatype's copy
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                             ! 0 on success and -1 on failure
END SUBROUTINE h5tcopy_f
```

Name: H5Tcreate

Signature:

hid_t `H5Tcreate`(*H5T_class_t* `class`, *size_t*`size`)

Purpose:

Creates a new datatype.

Description:

`H5Tcreate` creates a new datatype of the specified class with the specified number of bytes.

The following datatype classes are supported with this function:

- ◊ `H5T_COMPOUND`
- ◊ `H5T_OPAQUE`
- ◊ `H5T_ENUM`

Use `H5Tcopy` to create integer or floating-point datatypes.

The datatype identifier returned from this function should be released with `H5Tclose` or resource leaks will result.

Parameters:

H5T_class_t `class`	Class of datatype to create.
size_t `size`	The number of bytes in the datatype to create.

Returns:

Returns datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5tcreate_f

```
SUBROUTINE h5tcreate_f(class, size, type_id, hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(IN) :: class             ! Datatype class can be one of
                                           !    H5T_COMPOUND_F (6)
                                           !    H5T_ENUM_F     (8)
                                           !    H5T_OPAQUE_F   (9)
  INTEGER(SIZE_T), INTENT(IN) :: size      ! Size of the datatype
  INTEGER(HID_T), INTENT(OUT) :: type_id   ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5tcreate_f
```

Name: H5Tdecode

Signature:

hid_t `H5Tdecode` (*unsigned char* `*buf`)

Purpose:

Decode a binary object description of data type and return a new object handle.

Description:

Given an object description of data type in binary in a buffer, `H5Tdecode` reconstructs the HDF5 data type object and returns a new object handle for it. The binary description of the object is encoded by `H5Tencode`. User is responsible for passing in the right buffer.

Parameters:

unsigned char `*buf`	IN: Buffer for the data type object to be decoded.

Returns:

Returns an object ID(non-negative) if successful; otherwise returns a negative value.

Fortran90 Interface: h5tdecode_f

```
SUBROUTINE h5tdecode_f(buf, obj_id, hdferr)
  IMPLICIT NONE
  CHARACTER(LEN=*), INTENT(IN) :: buf  ! Data space object buffer to be decoded
  INTEGER(HID_T), INTENT(OUT) :: obj_id! Object ID
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5tdecode_f
```

Name: H5Tdetect_class

Signature:

htri_t `H5Tdetect_class`(*hid_t* `dtype_id`, *H5T_class_t*`dtype_class`)

Purpose:

Determines whether a datatype contains any datatypes of the given datatype class.

Description:

`H5Tdetect_class` determines whether the datatype specified in `dtype_id` contains any datatypes of the datatype class specified in `dtype_class`.

This function is useful primarily in recursively examining all the fields and/or base types of compound, array, and variable-length datatypes.

Valid class identifiers are as defined in `H5Tget_class`.

Parameters:

hid_t `dtype_id`	Datatype identifier.
H5T_class_t `dtype_class`	Datatype class.

Returns:

Returns `TRUE` or `FALSE` if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Tencode

Signature:

herr_t `H5Tencode`(*hid_t* `obj_id`, *unsigned char* *`buf`, *size_t* *`nalloc`)

Purpose:

Encode a data type object description into a binary buffer.

Description:

Given data type ID, `H5Tencode` converts a data type description into binary form in a buffer. Using this binary form in the buffer, a data type object can be reconstructed using `H5Tdecode` to return a new object handle (*hid_t*) for this data type.

A preliminary `H5Tencode` call can be made to find out the size of the buffer needed. This value is returned as `nalloc`. That value can then be assigned to `nalloc` for a second `H5Tencode` call, which will retrieve the actual encoded object.

If the library finds out `nalloc` is not big enough for the object, it simply returns the size of the buffer needed through `nalloc` without encoding the provided buffer.

Parameters:

hid_t `obj_id`	IN: Identifier of the object to be encoded.
unsigned char *`buf`	IN/OUT: Buffer for the object to be encoded into. If the provided buffer is NULL, only the size of buffer needed is returned through `nalloc`.
size_t *`nalloc`	IN: The size of the allocated buffer. OUT: The size of the buffer needed.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tencode_f

```
SUBROUTINE h5tencode_f(obj_id, buf, nalloc, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: obj_id ! Identifier of the object to be encoded
  CHARACTER(LEN=*), INTENT(OUT) :: buf ! Buffer object to be encoded into
  INTEGER(SIZE_T), INTENT(INOUT) :: nalloc
                                       ! The size of the allocated buffer
  INTEGER, INTENT(OUT) :: hdferr       ! Error code
                                       ! 0 on success and -1 on failure
END SUBROUTINE h5tencode_f
```

Name: H5Tenum_create
Signature:
hid_t `H5Tenum_create`(*hid_t* `parent_id`)
Purpose:
Creates a new enumeration datatype.
Description:
`H5Tenum_create` creates a new enumeration datatype based on the specified base datatype, `parent_id`, which must be an integer type.
Parameters:

hid_t `parent_id`	IN: Datatype identifier for the base datatype.

Returns:
Returns the datatype identifier for the new enumeration datatype if successful; otherwise returns a negative value.
Fortran90 Interface: h5tenum_create_f

```
SUBROUTINE h5tenum_create_f(parent_id, new_type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: parent_id     ! Datatype identifier for
                                              ! the  base datatype
  INTEGER(HID_T), INTENT(OUT) :: new_type_id  ! Datatype identifier for the
                                              ! new enumeration datatype
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
END SUBROUTINE h5tenum_create_f
```

Name: H5Tenum_insert

Signature:

herr_t `H5Tenum_insert(` *hid_t* `dtype_id`, *const char* `*name`, *void* `*value`)

Purpose:

Inserts a new enumeration datatype member.

Description:

`H5Tenum_insert` inserts a new enumeration datatype member into an enumeration datatype.

`dtype_id` is the enumeration datatype, `name` is the name of the new member, and `value` points to the value of the new member.

`name` and `value` must both be unique within `dtype_id`.

`value` points to data which is of the datatype defined when the enumeration datatype was created.

Parameters:

hid_t `dtype_id`	IN: Datatype identifier for the enumeration datatype.
const char `*name`	IN: Name of the new member.
void `*value`	IN: Pointer to the value of the new member.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tenum_insert_f

```
SUBROUTINE h5tenum_insert_f(type_id,  name, value, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of the new member
  INTEGER, INTENT(IN) :: value          ! Value of the new member
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tenum_insert_f
```

Name: H5Tenum_nameof

Signature:

herr_t `H5Tenum_nameof(` *hid_t* `dtype_id`, *void* `*value`, *char* `*name`, *size_t* `size`)

Purpose:

Returns the symbol name corresponding to a specified member of an enumeration datatype.

Description:

`H5Tenum_nameof` finds the symbol name that corresponds to the specified `value` of the enumeration datatype `dtype_id`.

At most `size` characters of the symbol name are copied into the `name` buffer. If the entire symbol name and null terminator do not fit in the `name` buffer, then as many characters as possible are copied (not null terminated) and the function fails.

Parameters:

hid_t `dtype_id`	IN: Enumeration datatype identifier.
void `*value`	IN: Value of the enumeration datatype.
char `*name`	OUT: Buffer for output of the symbol name.
size_t `size`	IN: Anticipated size of the symbol name, in bytes (characters).

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value and, if `size` allows it, the first character of `name` is set to `NULL`.

Fortran90 Interface: h5tenum_nameof_f

```
SUBROUTINE h5tenum_nameof_f(type_id, value, namelen, name, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  INTEGER, INTENT(IN) :: value           ! Value of the enumeration datatype
  INTEGER(SIZE_T), INTENT(IN) :: namelen ! Length of the name
  CHARACTER(LEN=*), INTENT(OUT) :: name  ! Name of the enumeration datatype
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5tenum_nameof_f
```

Name: H5Tenum_valueof

Signature:

herr_t `H5Tenum_valueof`(*hid_t* `dtype_id`, *char* *`name`, *void* *`value`)

Purpose:

Returns the value corresponding to a specified member of an enumeration datatype.

Description:

`H5Tenum_valueof` finds the value that corresponds to the specified `name` of the enumeration datatype `dtype_id`.

The `value` argument should be at least as large as the value of `H5Tget_size(type)` in order to hold the result.

Parameters:

hid_t `dtype_id`	IN: Enumeration datatype identifier.
const char *`name`	IN: Symbol name of the enumeration datatype.
void *`value`	OUT: Buffer for output of the value of the enumeration datatype.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tenum_valueof_f

```
SUBROUTINE h5tenum_valueof_f(type_id,  name, value, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  CHARACTER(LEN=*), INTENT(IN) :: name  ! Name of the enumeration datatype
  INTEGER, INTENT(OUT) :: value         ! Value of the enumeration datatype
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tenum_valueof_f
```

Name: H5Tequal

Signature:

htri_t `H5Tequal`(*hid_t* `dtype_id1`, *hid_t* `dtype_id2`)

Purpose:

Determines whether two datatype identifiers refer to the same datatype.

Description:

`H5Tequal` determines whether two datatype identifiers refer to the same datatype.

Parameters:

hid_t `dtype_id1`	Identifier of datatype to compare.
hid_t `dtype_id2`	Identifier of datatype to compare.

Returns:

When successful, returns a positive value, for `TRUE`, if the datatype identifiers refer to the same datatype, or `0` (zero), for `FALSE`. Otherwise returns a negative value.

Fortran90 Interface: h5tequal_f

```
SUBROUTINE h5tequal_f(type1_id, type2_id, flag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type1_id ! Datatype identifier
  INTEGER(HID_T), INTENT(IN) :: type2_id ! Datatype identifier
  LOGICAL, INTENT(OUT) :: flag           ! TRUE/FALSE flag to indicate
                                         ! if two datatypes are equal
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5tequal_f
```

Name: H5Tfind

Signature:

H5T_conv_t `H5Tfind`(*hid_t* `src_id`, *hid_t* `dst_id`, *H5T_cdata_t* **`pcdata`)

Purpose:

Finds a conversion function.

Description:

`H5Tfind` finds a conversion function that can handle a conversion from type `src_id` to type `dst_id`. The `pcdata` argument is a pointer to a pointer to type conversion data which was created and initialized by the soft type conversion function of this path when the conversion function was installed on the path.

Parameters:

hid_t `src_id`	IN: Identifier for the source datatype.
hid_t `dst_id`	IN: Identifier for the destination datatype.
H5T_cdata_t **`pcdata`	OUT: Pointer to type conversion data.

Returns:

Returns a pointer to a suitable conversion function if successful. Otherwise returns NULL.

Fortran90 Interface:

None.

Name: H5Tget_array_dims

Signatures:

int `H5Tget_array_dims(` *hid_t* `adtype_id`, *hsize_t* `dims[]`, *int* `perm[]`) [1]

int `H5Tget_array_dims(` *hid_t* `adtype_id`, *hsize_t* `dims[]`) [2]

Purpose:

Retrieves sizes of array dimensions.

Description:

`H5Tget_array_dims` is a macro that is mapped to either `H5Tget_array_dims1` or `H5Tget_array_dims2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. For example:

◊ The `H5Tget_array_dims` macro will be mapped to `H5Tget_array_dims1` and will use the `H5Tget_array_dims1` syntax (first signature above) if an application is coded for HDF5 Release 1.6.*x*.

◊ The `H5Tget_array_dims` macro mapped to `H5Tget_array_dims2` and will use the `H5Tget_array_dims2` syntax (second signature above) if an application is coded for HDF5 Release 1.8.*x*.

Macro use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Tget_array_dims` is mapped to the most recent version of the function, currently `H5Tget_array_dims2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Tget_array_dims` will be mapped to `H5Tget_array_dims1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	**`H5Tget_array_dims` mapping**
Global settings	
No compatibility flag	`H5Tget_array_dims2`
Enable deprecated symbols	`H5Tget_array_dims2`
Disable deprecated symbols	`H5Tget_array_dims2`
Emulate Release 1.6 interface	`H5Tget_array_dims1`
Function-level macros	
`H5Tget_array_dims_vers = 2`	`H5Tget_array_dims2`
`H5Tget_array_dims_vers = 1`	`H5Tget_array_dims1`

Interface history: Signature [1] above is the original `H5Tget_array_dims` interface and the only interface available prior to HDF5 Release 1.8.0. This signature and the corresponding function are now deprecated but will remain directly callable as `H5Tget_array_dims1`.

Signature [2] above was introduced with HDF5 Release 1.8.0 and is the recommended and default interface. It is directly callable as `H5Tget_array_dims2`.

See “API Compatibility Macros in HDF5” for circumstances under which either of these functions might not be available in an installed instance of the HDF5 Library.

Fortran90 Interface: h5tarray_create_f

```
SUBROUTINE h5tarray_create_f(base_id, rank, dims, type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: base_id   ! Identifier of array base datatype
  INTEGER, INTENT(IN)        ::  rank     ! Rank of the array
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(IN) :: dims
                                          ! Sizes of each array dimension
  INTEGER(HID_T), INTENT(OUT) :: type_id  ! Identifier of the array datatype
  INTEGER, INTENT(OUT)        :: hdferr   ! Error code
END SUBROUTINE h5tarray_create_f
```

History:

Release	**C**
1.8.0	The function `H5Tget_array_dims` renamed to `H5Tget_array_dims1` and deprecated in this release. The macro `H5Tget_array_dims` and the function `H5Tget_array_dims2` introduced in this release.

Name: H5Tget_array_dims1

Signature:

int `H5Tget_array_dims1`(*hid_t* `adtype_id`, *hsize_t* `dims[]`, *int* `perm[]`)

Purpose:

Retrieves sizes of array dimensions.

Notice:

This function is renamed from `H5Tget_array_dims` and deprecated in favor of the function `H5Tget_array_dims2` or the new macro `H5Tget_array_dims`.

Description:

`H5Tget_array_dims1` returns the sizes of the dimensions and the dimension permutations of the specified array datatype object.

The sizes of the dimensions are returned in the array `dims`.

The parameter `perm` is not used.

Parameters:

hid_t `adtype_id`	IN: Datatype identifier of array object.
hsize_t `dims[]`	OUT: Sizes of array dimensions.
int `perm[]`	OUT: Dimension permutations. *(This parameter is not used.)*

Returns:

Returns the non-negative number of dimensions of the array type if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_array_dims_f

```
SUBROUTINE h5tget_array_dims_f(type_id, dims, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id      ! Identifier of the array datatype
  INTEGER(HSIZE_T), DIMENSION(*), INTENT(OUT) ::  dims
                                             ! Buffer to store array datatype
  INTEGER, INTENT(OUT)  :: hdferr            ! Error code
END SUBROUTINE h5tget_array_dims_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.
1.8.0	The function `H5Tget_array_dims` renamed to `H5Tget_array_dims1` and deprecated in this release.

Name: H5Tget_array_dims2

Signature:

int `H5Tget_array_dims2`(*hid_t* `adtype_id`, *hsize_t* `dims[]`)

Purpose:

Retrieves sizes of array dimensions.

Description:

`H5Tget_array_dims2` returns the sizes of the dimensions of the specified array datatype object.

The sizes of the dimensions are returned in the array `dims`.

Parameters:

hid_t `adtype_id`	IN: Datatype identifier of array object.
hsize_t `dims[]`	OUT: Sizes of array dimensions.

Returns:

Returns the non-negative number of dimensions of the array type if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Tget_array_ndims

Signature:

int `H5Tget_array_ndims(` *hid_t* `adtype_id`)

Purpose:

Returns the rank of an array datatype.

Description:

`H5Tget_array_ndims` returns the rank, the number of dimensions, of an array datatype object.

Parameters:

hid_t `adtype_id`	IN: Datatype identifier of array object.

Returns:

Returns the rank of the array if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_array_ndims_f

```
SUBROUTINE h5tget_array_ndims_f(type_id, ndims, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Identifier of the array datatype
  INTEGER, INTENT(OUT)       ::  ndims   ! Number of array dimensions
  INTEGER, INTENT(OUT)       :: hdferr   ! Error code
END SUBROUTINE h5tget_array_ndims_f
```

History:

Release	**C**
1.4.0	Function introduced in this release.

Name: H5Tget_class

Signature:

H5T_class_t `H5Tget_class(` *hid_t* `dtype_id`)

Purpose:

Returns the datatype class identifier.

Description:

`H5Tget_class` returns the datatype class identifier.

Valid class identifiers, as defined in `H5Tpublic.h`, are:

◊ `H5T_INTEGER`
◊ `H5T_FLOAT`
◊ `H5T_STRING`
◊ `H5T_BITFIELD`
◊ `H5T_OPAQUE`
◊ `H5T_COMPOUND`
◊ `H5T_REFERENCE`
◊ `H5T_ENUM`
◊ `H5T_VLEN`
◊ `H5T_ARRAY`

Note that the library returns `H5T_STRING` for both fixed-length and variable-length strings.

Unsupported datatype: The time datatype class, `H5T_TIME`, is not supported. If `H5T_TIME` is used, the resulting data will be readable and modifiable only on the originating computing platform; it will not be portable to other platforms.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns datatype class identifier if successful; otherwise `H5T_NO_CLASS` (-1).

Fortran90 Interface: h5tget_class_f

```
SUBROUTINE h5tget_class_f(type_id, class, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  INTEGER, INTENT(OUT) :: class          ! Datatype class, possible values are:
                                         !    H5T_NO_CLASS_F
                                         !    H5T_INTEGER_F
                                         !    H5T_FLOAT_F

                                         !    H5T_STRING_F
                                         !    H5T_BITFIELD_F
                                         !    H5T_OPAQUE_F
                                         !    H5T_COMPOUND_F
                                         !    H5T_REFERENCE_F
                                         !    H5T_ENUM_F
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5tget_class_f
```

Name: H5Tget_create_plist

Signature:

hid_t `H5Tget_create_plist(` *hid_t* `dtype_id`)

Purpose:

Returns a copy of a datatype creation property list.

Description:

`H5Tget_create_plist` returns a property list identifier for the datatype creation property list associated with the datatype specified by `dtype_id`.

The creation property list identifier should be released with `H5Pclose`.

Parameter:

hid_t `dtype_id`	IN: Datatype identifier.

Returns:

Returns a datatype property list identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_create_plist_f

```
SUBROUTINE h5tget_create_plist_f(dtype_id, dtpl_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dtype_id ! Datatype identifier
  INTEGER(HID_T), INTENT(OUT) :: dtpl_id ! Datatype property list identifier.
  INTEGER, INTENT(OUT) :: hdferr         ! Error code:
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5tget_create_plist_f
```

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Tget_cset

Signature:

H5T_cset_t `H5Tget_cset(` *hid_t* `dtype_id`)

Purpose:

Retrieves the character set type of a string datatype.

Description:

`H5Tget_cset` retrieves the character set type of a string datatype. Valid character set types are:

`H5T_CSET_ASCII`	(`0`)	Character set is US ASCII.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns a valid character set type if successful; otherwise `H5T_CSET_ERROR` (-1).

Fortran90 Interface: h5tget_cset_f

```
SUBROUTINE h5tget_cset_f(type_id, cset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: cset          ! Character set type of a string
                                        ! datatype
                                        ! Possible values of padding type are:
                                        !    H5T_CSET_ASCII_F = 0
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_cset_f
```

Name: H5Tget_ebias

Signature:

size_t `H5Tget_ebias`(*hid_t* `dtype_id`)

Purpose:

Retrieves the exponent bias of a floating-point type.

Description:

`H5Tget_ebias` retrieves the exponent bias of a floating-point type.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns the bias if successful; otherwise 0.

Fortran90 Interface: h5tget_ebias_f

```
SUBROUTINE h5tget_ebias_f(type_id, ebias, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER(SIZE_T), INTENT(OUT) :: ebias ! Datatype exponent bias
                                        ! of a floating-point type
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_ebias_f
```

Name: H5Tget_fields

Signature:

herr_t `H5Tget_fields`(*hid_t* `dtype_id`, *size_t* *`spos`, *size_t* *`epos`, *size_t* *`esize`, *size_t* *`mpos`, *size_t* *`msize`)

Purpose:

Retrieves floating point datatype bit field information.

Description:

`H5Tget_fields` retrieves information about the locations of the various bit fields of a floating point datatype. The field positions are bit positions in the significant region of the datatype. Bits are numbered with the least significant bit number zero. Any (or even all) of the arguments can be null pointers.

Parameters:

hid_t `dtype_id`	IN: Identifier of datatype to query.
size_t *`spos`	OUT: Pointer to location to return floating-point sign bit.
size_t *`epos`	OUT: Pointer to location to return exponent bit-position.
size_t *`esize`	OUT: Pointer to location to return size of exponent in bits.
size_t *`mpos`	OUT: Pointer to location to return mantissa bit-position.
size_t *`msize`	OUT: Pointer to location to return size of mantissa in bits.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_fields_f

```
SUBROUTINE h5tget_fields_f(type_id, spos, epos, esize, mpos, msize, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER(SIZE_T), INTENT(OUT) :: spos  ! sign bit-position
  INTEGER(SIZE_T), INTENT(OUT) :: epos  ! exponent bit-position
  INTEGER(SIZE_T), INTENT(OUT) :: esize ! size of exponent in bits
  INTEGER(SIZE_T), INTENT(OUT) :: mpos  ! mantissa bit-position
  INTEGER(SIZE_T), INTENT(OUT) :: msize ! size of mantissa in bits
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_fields_f
```

Name: H5Tget_inpad

Signature:

H5T_pad_t `H5Tget_inpad`(*hid_t* `dtype_id`)

Purpose:

Retrieves the internal padding type for unused bits in floating-point datatypes.

Description:

`H5Tget_inpad` retrieves the internal padding type for unused bits in floating-point datatypes. Valid padding types are:

H5T_PAD_ZERO (`0`)
: Set background to zeros.

H5T_PAD_ONE (`1`)
: Set background to ones.

H5T_PAD_BACKGROUND (`2`)
: Leave background alone.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns a valid padding type if successful; otherwise `H5T_PAD_ERROR` (-1).

Fortran90 Interface: h5tget_inpad_f

```
SUBROUTINE h5tget_inpad_f(type_id, padtype, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: padtype       ! Padding type for unused bits
                                        ! in floating-point datatypes
                                        ! Possible values of padding type are:
                                        !     H5T_PAD_ZERO_F = 0
                                        !     H5T_PAD_ONE_F = 1
                                        !     H5T_PAD_BACKGROUND_F = 2
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_inpad_f
```

Name: H5Tget_member_class

Signature:

H5T_class_t `H5Tget_member_class`(*hid_t* `cdtype_id`, *unsigned* `member_no`)

Purpose:

Returns datatype class of compound datatype member.

Description:

Given a compound datatype, `cdtype_id`, the function `H5Tget_member_class` returns the datatype class of the compound datatype member specified by `member_no`.

Valid class identifiers are as defined in `H5Tget_class`.

Parameters:

hid_t `cdtype_id`	IN: Datatype identifier of compound object.
unsigned `member_no`	IN: Compound object member number.

Returns:

Returns the datatype class, a non-negative value, if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_member_class_f

```
SUBROUTINE h5tget_member_class_f(type_id, member_no, class, hdferr)
  INTEGER(HID_T), INTENT(IN) :: type_id       ! Datatype identifier
  INTEGER, INTENT(IN) :: member_no            ! Member number
  INTEGER, INTENT(OUT) :: class               ! Member class
  INTEGER, INTENT(OUT) :: hdferr              ! Error code
END SUBROUTINE h5tget_member_class_f
```

History:

Release	**C**
1.6.4	`membno` parameter type changed to *unsigned*.

Name: H5Tget_member_index

Signature:

int `H5Tget_member_index`(*hid_t* `dtype_id`, *const char* * `field_name`)

Purpose:

Retrieves the index of a compound or enumeration datatype member.

Description:

`H5Tget_member_index` retrieves the index of a field of a compound datatype or an element of an enumeration datatype.

The name of the target field or element is specified in `field_name`.

Fields are stored in no particular order with index values of 0 through *N*-1, where *N* is the value returned by `H5Tget_nmembers`.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.
const char * `field_name`	Name of the field or member whose index is to be retrieved.

Returns:

Returns a valid field or member index if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_member_index_f

```
SUBROUTINE h5tget_member_index_f(type_id, name, index, hdferr)
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Member name
  INTEGER, INTENT(OUT) :: index          ! Member index
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5tget_member_index_f
```

History:

Release	**C**	**Fortran90**
1.4.5		Function introduced in this release.
1.4.4	Function introduced in this release.	

Name: H5Tget_member_name

Signature:

char * `H5Tget_member_name`(*hid_t* `dtype_id`, *unsigned* `field_idx`)

Purpose:

Retrieves the name of a compound or enumeration datatype member.

Description:

`H5Tget_member_name` retrieves the name of a field of a compound datatype or an element of an enumeration datatype.

The index of the target field or element is specified in `field_idx`. Compound datatype fields and enumeration datatype elements are stored in no particular order with index values of 0 through *N*-1, where *N* is the value returned by `H5Tget_nmembers`.

A buffer to receive the name of the field is allocated with `malloc()` and the caller is responsible for freeing the memory used.

Parameters:

hid_t `tdype_id`	Identifier of datatype to query.
unsigned `field_idx`	Zero-based index of the field or element whose name is to be retrieved.

Returns:

Returns a valid pointer to a string allocated with `malloc()` if successful; otherwise returns NULL.

Fortran90 Interface: h5tget_member_name_f

```
SUBROUTINE h5tget_member_name_f(type_id,index, member_name,  namelen, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id        ! Datatype identifier
  INTEGER, INTENT(IN) :: index                 ! Field index (0-based) of
                                               ! the field name to retrieve
  CHARACTER(LEN=*), INTENT(OUT) :: member_name ! Name of a field of
                                               ! a compound datatype
  INTEGER, INTENT(OUT) :: namelen              ! Length of the name
  INTEGER, INTENT(OUT) :: hdferr               ! Error code
END SUBROUTINE h5tget_member_name_f
```

History:

Release	**C**
1.6.4	`membno` parameter type changed to *unsigned*.

Name: H5Tget_member_offset

Signature:

size_t `H5Tget_member_offset(` *hid_t* `dtype_id`, *unsigned* `memb_no`)

Purpose:

Retrieves the offset of a field of a compound datatype.

Description:

`H5Tget_member_offset` retrieves the byte offset of the beginning of a field within a compound datatype with respect to the beginning of the compound data type datum.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.
unsigned `memb_no`	Number of the field whose offset is requested.

Returns:

Returns the byte offset of the field if successful; otherwise returns `0` (zero). Note that zero is a valid offset and that this function will fail only if a call to `H5Tget_member_class()` fails with the same arguments.

Fortran90 Interface: h5tget_member_offset_f

```
SUBROUTINE h5tget_member_offset_f(type_id, member_no, offset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id    ! Datatype identifier
  INTEGER, INTENT(IN) :: member_no         ! Number of the field
                                           ! whose offset is requested
  INTEGER(SIZE_T), INTENT(OUT) :: offset   ! Byte offset of the the
                                           ! beginning of the field
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
END SUBROUTINE h5tget_member_offset_f
```

History:

Release	**C**
1.6.4	`membno` parameter type changed to *unsigned*.

Name: H5Tget_member_type

Signature:

hid_t `H5Tget_member_type(` *hid_t* `dtype_id`, *unsigned* `field_idx`)

Purpose:

Returns the datatype of the specified member.

Description:

`H5Tget_member_type` returns the datatype of the specified member. The caller should invoke H5Tclose() to release resources associated with the type.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.
unsigned `field_idx`	Field index (0-based) of the field type to retrieve.

Returns:

Returns the identifier of a copy of the datatype of the field if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_member_type_f

```
SUBROUTINE h5tget_member_type_f(type_id,  field_idx, datatype, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id   ! Datatype identifier
  INTEGER, INTENT(IN) :: field_idx        ! Field index (0-based) of the
                                          ! field type to retrieve
  INTEGER(HID_T), INTENT(OUT) :: datatype ! Identifier of a copy of
                                          ! the datatype of the field
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
END SUBROUTINE h5tget_member_type_f
```

History:

Release	**C**
1.6.4	`membno` parameter type changed to *unsigned*.

Name: H5Tget_member_value

Signature:

herr_t `H5Tget_member_value`(*hid_t* `dtype_id` *unsigned* `memb_no`, *void* *`value`)

Purpose:

Returns the value of an enumeration datatype member.

Description:

`H5Tget_member_value` returns the value of the enumeration datatype member `memb_no`.

The member value is returned in a user-supplied buffer pointed to by `value`.

Parameters:

hid_t `dtype_id`	IN: Datatype identifier for the enumeration datatype.
unsigned `memb_no`	IN: Number of the enumeration datatype member.
void *`value`	OUT: Pointer to a buffer for output of the value of the enumeration datatype member.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_member_value_f

```
SUBROUTINE h5tget_member_value_f(type_id,  member_no, value, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(IN) :: member_no      ! Number of the enumeration
                                        ! datatype member
  INTEGER, INTENT(OUT) :: value         ! Value of the enumeration datatype
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_member_value_f
```

History:

Release	**C**
1.6.4	`membno` parameter type changed to *unsigned*.

Name: H5Tget_native_type
Signature:

hid_t `H5Tget_native_type(` *hid_t* `dtype_id`, *H5T_direction_t* `direction`)

Purpose:

Returns the native datatype of a specified datatype.

Description:

`H5Tget_native_type` returns the equivalent native datatype for the datatype specified in `dtype_id`.

`H5Tget_native_type` is a high-level function designed primarily to facilitate use of the `H5Dread` function, for which users otherwise must undertake a multi-step process to determine the native datatype of a dataset prior to reading it into memory. It can be used not only to determine the native datatype for atomic datatypes, but also to determine the native datatypes of the individual components of a compound datatype, an enumerated datatype, an array datatype, or a variable-length datatype. For example, if `dtype_id` is a compound datatype, the returned datatype identifier will be for a similar compound datatype with each element converted to the corresponding native datatype.

`H5Tget_native_type` selects the matching native datatype from the following list:

```
H5T_NATIVE_CHAR
H5T_NATIVE_SHORT
H5T_NATIVE_INT
H5T_NATIVE_LONG
H5T_NATIVE_LLONG

H5T_NATIVE_UCHAR
H5T_NATIVE_USHORT
H5T_NATIVE_UINT
H5T_NATIVE_ULONG
H5T_NATIVE_ULLONG

H5T_NATIVE_FLOAT
H5T_NATIVE_DOUBLE
H5T_NATIVE_LDOUBLE
```

The `direction` parameter indicates the order in which the library searches for a native datatype match. Valid values for `direction` are as follows:

`H5T_DIR_ASCEND`	Searches the above list in ascending size of the datatype, i.e., from top to bottom. (Default)
`H5T_DIR_DESCEND`	Searches the above list in descending size of the datatype, i.e., from bottom to top.

`H5Tget_native_type` is designed primarily for use with integer and floating point datatypes. String, time, bitfield, opaque, and reference datatypes are returned as a copy of `dtype_id`.

The identifier returned by `H5Tget_native_type` should eventually be closed by calling `H5Tclose` to release resources.

Parameters:

hid_t `dtype_id`	Datatype identifier for the dataset datatype.
H5T_direction_t `direction`	Direction of search.

Returns:

Returns the native datatype identifier for the specified dataset datatype if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_native_type_f

```
SUBROUTINE h5tget_native_type_f(dtype_id, direction, native_dtype_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dtype_id
                                    ! Datatype identifier
  INTEGER, INTENT(IN) :: direction  ! Direction of search:
                                    ! H5T_DIR_ASCEND_F = 1 in inscendent order
                                    ! H5T_DIR_DESCEND_F= 2 in descendent order
  INTEGER(HID_T), INTENT(OUT) :: native_dtype_id
                                    ! The native datatype identifier
  INTEGER, INTENT(OUT) :: hdferr    ! Error code:
                                    ! 0 on success and -1 on failure
END SUBROUTINE h5tget_native_type_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Tget_nmembers

Signature:

int `H5Tget_nmembers`(*hid_t* `dtype_id`)

Purpose:

Retrieves the number of elements in a compound or enumeration datatype.

Description:

`H5Tget_nmembers` retrieves the number of fields in a compound datatype or the number of members of an enumeration datatype.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns the number of elements if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_nmembers_f

```
SUBROUTINE h5tget_nmembers_f(type_id, num_members, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: num_members   ! Number of fields in a
                                        ! compound datatype
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_nmembers_f
```

Name: H5Tget_norm

Signature:

H5T_norm_t `H5Tget_norm`(*hid_t* `dtype_id`)

Purpose:

Retrieves mantissa normalization of a floating-point datatype.

Description:

`H5Tget_norm` retrieves the mantissa normalization of a floating-point datatype. Valid normalization types are:

H5T_NORM_IMPLIED (`0`)
: MSB of mantissa is not stored, always 1

H5T_NORM_MSBSET (`1`)
: MSB of mantissa is always 1

H5T_NORM_NONE (`2`)
: Mantissa is not normalized

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns a valid normalization type if successful; otherwise `H5T_NORM_ERROR` (-1).

Fortran90 Interface: h5tget_norm_f

```
SUBROUTINE h5tget_norm_f(type_id, norm, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id
                                ! Datatype identifier
  INTEGER, INTENT(OUT) :: norm   ! Mantissa normalization of a
                                ! floating-point datatype
                                ! Valid normalization types are:
                                !    H5T_NORM_IMPLIED_F(0)
                                !        MSB of mantissa is not
                                !        stored, always 1
                                !    H5T_NORM_MSBSET_F(1)
                                !        MSB of mantissa is always 1
                                !    H5T_NORM_NONE_F(2)
                                !        Mantissa is not normalized
  INTEGER, INTENT(OUT) :: hdferr ! Error code
END SUBROUTINE h5tget_norm_f
```

Name: H5Tget_offset

Signature:

int `H5Tget_offset(` *hid_t* `dtype_id`)

Purpose:

Retrieves the bit offset of the first significant bit.

Description:

`H5Tget_offset` retrieves the bit offset of the first significant bit. The significant bits of an atomic datum can be offset from the beginning of the memory for that datum by an amount of padding. The `offset' property specifies the number of bits of padding that appear to the "right of" the value. That is, if we have a 32-bit datum with 16-bits of precision having the value 0x1122 then it will be laid out in memory as (from small byte address toward larger byte addresses):

Byte Position	Big-Endian Offset=0	Big-Endian Offset=16	Little-Endian Offset=0	Little-Endian Offset=16
0:	[pad]	[0x11]	[0x22]	[pad]
1:	[pad]	[0x22]	[0x11]	[pad]
2:	[0x11]	[pad]	[pad]	[0x22]
3:	[0x22]	[pad]	[pad]	[0x11]

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns an offset value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_offset_f

```
SUBROUTINE h5tget_offset_f(type_id, offset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  INTEGER(SIZE_T), INTENT(OUT) :: offset ! Datatype bit offset of the
                                         ! first significant bit
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5tget_offset_f
```

Name: H5Tget_order

Signature:

H5T_order_t `H5Tget_order(` *hid_t* `dtype_id`)

Purpose:

Returns the byte order of an atomic datatype.

Description:

`H5Tget_order` returns the byte order of an atomic datatype.

Possible return values are:

`H5T_ORDER_LE` (0)
: Little-endian byte order

`H5T_ORDER_BE` (1)
: Big-endian byte order

`H5T_ORDER_VAX` (2)
: VAX mixed byte order (not currently supported)

`H5T_ORDER_NONE` (3)
: No particular order (fixed-length strings, object and region references)

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns a byte order constant if successful; otherwise `H5T_ORDER_ERROR` (-1).

Fortran90 Interface: h5tget_order_f

```
SUBROUTINE h5tget_order_f(type_id, order, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: order         ! Datatype byte order
                                        ! Possible values are:
                                        !    H5T_ORDER_LE_F
                                        !    H5T_ORDER_BE_F
                                        !    H5T_ORDER_VAX_F
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5tget_order_f
```

Name: H5Tget_pad

Signature:

herr_t `H5Tget_pad`(*hid_t* `dtype_id`, *H5T_pad_t* * `lsb`, *H5T_pad_t* * `msb`)

Purpose:

Retrieves the padding type of the least and most-significant bit padding.

Description:

`H5Tget_pad` retrieves the padding type of the least and most-significant bit padding. Valid types are:

H5T_PAD_ZERO (`0`)
: Set background to zeros.

H5T_PAD_ONE (`1`)
: Set background to ones.

H5T_PAD_BACKGROUND (`2`)
: Leave background alone.

Parameters:

hid_t `dtype_id`	IN: Identifier of datatype to query.
H5T_pad_t * `lsb`	OUT: Pointer to location to return least-significant bit padding type.
H5T_pad_t * `msb`	OUT: Pointer to location to return most-significant bit padding type.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_pad_f

```
SUBROUTINE h5tget_pad_f(type_id, lsbpad, msbpad, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: lsbpad        ! Padding type of the
                                        ! least significant bit
  INTEGER, INTENT(OUT) :: msbpad        ! Padding type of the
                                        ! most significant bit
                                        ! Possible values of
                                        ! padding type are:
                                        !    H5T_PAD_ZERO_F = 0
                                        !    H5T_PAD_ONE_F = 1
                                        !    H5T_PAD_BACKGROUND_F = 2
                                        !    H5T_PAD_ERROR_F = -1
                                        !    H5T_PAD_NPAD_F = 3
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_pad_f
```

Name: H5Tget_precision

Signature:

size_t `H5Tget_precision`(*hid_t* `dtype_id`)

Purpose:

Returns the precision of an atomic datatype.

Description:

`H5Tget_precision` returns the precision of an atomic datatype. The precision is the number of significant bits which, unless padding is present, is 8 times larger than the value returned by `H5Tget_size`.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns the number of significant bits if successful; otherwise 0.

Fortran90 Interface: h5tget_precision_f

```
SUBROUTINE h5tget_precision_f(type_id, precision, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id     ! Datatype identifier
  INTEGER(SIZE_T), INTENT(OUT) :: precision ! Datatype precision
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
END SUBROUTINE h5tget_precision_f
```

Name: H5Tget_sign

Signature:

H5T_sign_t `H5Tget_sign(` *hid_t* `dtype_id`)

Purpose:

Retrieves the sign type for an integer type.

Description:

`H5Tget_sign` retrieves the sign type for an integer type. Valid types are:

H5T_SGN_NONE (`0`)
: Unsigned integer type.

H5T_SGN_2 (`1`)
: Two's complement signed integer type.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns a valid sign type if successful; otherwise `H5T_SGN_ERROR` (-1).

Fortran90 Interface: h5tget_sign_f

```
SUBROUTINE h5tget_sign_f(type_id, sign, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  INTEGER, INTENT(OUT) :: sign           ! Sign type for an integer type
                                         ! Possible values are:
                                         !    Unsigned integer type
                                         !        H5T_SGN_NONE_F = 0
                                         !    Two's complement signed
                                         !        integer type
                                         !        H5T_SGN_2_F = 1
                                         !    or error value
                                         !         H5T_SGN_ERROR_F = -1
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
END SUBROUTINE h5tget_sign_f
```

Name: H5Tget_size

Signature:

size_t `H5Tget_size`(*hid_t* `dtype_id`)

Purpose:

Returns the size of a datatype.

Description:

`H5Tget_size` returns the size of a datatype in bytes.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns the size of the datatype in bytes if successful; otherwise 0.

Fortran90 Interface: h5tget_size_f

```
SUBROUTINE h5tget_size_f(type_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER(SIZE_T), INTENT(OUT) :: size  ! Datatype size
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
                                        ! 0 on success and -1 on failure
END SUBROUTINE h5tget_size_f
```

Name: H5Tget_strpad

Signature:

H5T_str_t `H5Tget_strpad(` *hid_t* `dtype_id`)

Purpose:

Retrieves the storage mechanism for a string datatype.

Description:

`H5Tget_strpad` retrieves the storage mechanism for a string datatype, as defined in `H5Tset_strpad`.

Parameters:

hid_t `dtype_id`	Identifier of datatype to query.

Returns:

Returns a valid string storage mechanism if successful; otherwise `H5T_STR_ERROR` (-1).

Fortran90 Interface: h5tget_strpad_f

```
SUBROUTINE h5tget_strpad_f(type_id, strpad, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id
                                  ! Datatype identifier
  INTEGER, INTENT(OUT) :: strpad  ! String padding method for a string datatype
                                  ! Possible values of padding type are:
                                  !    Pad with zeros (as C does):
                                  !        H5T_STR_NULLPAD_F(0)
                                  !    Pad with spaces (as FORTRAN does):
                                  !        H5T_STR_SPACEPAD_F(1)
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
                                  ! 0 on success and -1 on failure
END SUBROUTINE h5tget_strpad_f
```

Name: H5Tget_super

Signature:

hid_t `H5Tget_super(` *hid_t* `dtype_id`)

Purpose:

Returns the base datatype from which a datatype is derived.

Description:

`H5Tget_super` returns the base datatype from which the datatype `dtype_id` is derived.

In the case of an enumeration type, the return value is an integer type.

Parameters:

hid_t `dtype_id`	Datatype identifier for the derived datatype.

Returns:

Returns the datatype identifier for the base datatype if successful; otherwise returns a negative value.

Fortran90 Interface: h5tget_super_f

```
SUBROUTINE h5tget_super_f(type_id, base_type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  INTEGER(HID_T), INTENT(OUT) :: type_id ! Base datatype identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5tget_super_f
```

Name: H5Tget_tag

Signature:

char *`H5Tget_tag`(*hid_t* `dtype_id`)

Purpose:

Gets the tag associated with an opaque datatype.

Description:

`H5Tget_tag` returns the tag associated with the opaque datatype `dtype_id`.

The tag is returned via a pointer to an allocated string, which the caller must free.

Parameters:

hid_t `dtype_id`	Datatype identifier for the opaque datatype.

Returns:

Returns a pointer to an allocated string if successful; otherwise returns `NULL`.

Fortran90 Interface: h5tget_tag_f

```
SUBROUTINE h5tget_tag_f(type_id, tag,taglen, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  CHARACTER(LEN=*), INTENT(OUT) :: tag   ! Unique ASCII string with which the
                                         ! opaque datatype is to be tagged
  INTEGER, INTENT(OUT) :: taglen         ! Length of tag
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5tget_tag_f
```

Name: H5Tinsert

Signature:

herr_t `H5Tinsert`(*hid_t* `dtype_id`, *const char* * `name`, *size_t* `offset`, *hid_t* `field_id`)

Purpose:

Adds a new member to a compound datatype.

Description:

`H5Tinsert` adds another member to the compound datatype `dtype_id`. The new member has a `name` which must be unique within the compound datatype. The `offset` argument defines the start of the member in an instance of the compound datatype, and `field_id` is the datatype identifier of the new member.

Note: Members of a compound datatype do not have to be atomic datatypes; a compound datatype can have a member which is a compound datatype.

Parameters:

hid_t `dtype_id`	Identifier of compound datatype to modify.
const char * `name`	Name of the field to insert.
size_t `offset`	Offset in memory structure of the field to insert.
hid_t `field_id`	Datatype identifier of the field to insert.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tinsert_f

```
SUBROUTINE h5tinsert_f(type_id,  name, offset, field_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Name of the field to insert
  INTEGER(SIZE_T), INTENT(IN) :: offset  ! Offset in memory structure
                                         ! of the field to insert
  INTEGER(HID_T), INTENT(IN) :: field_id ! Datatype identifier of the
                                         ! new member
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5tinsert_f
```

Name: H5Tis_variable_str

Signature:

htri_t `H5Tis_variable_str(` *hid_t* `dtype_id )`

Purpose:

Determines whether datatype is a variable-length string.

Description:

`H5Tis_variable_str` determines whether the datatype identified in `dtype_id` is a variable-length string.

This function can be used to distinguish between fixed and variable-length string datatypes.

Parameters:

hid_t `dtype_id`	Datatype identifier.

Returns:

Returns `TRUE` or `FALSE` if successful; otherwise returns a negative value.

Fortran90 Interface: h5tis_variable_str_f

```
SUBROUTINE h5tis_variable_str_f(type_id, status, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id   ! Datatype identifier
  LOGICAL, INTENT(OUT)       :: status    ! Logical flag:
                                          !    .TRUE. if datatype is a
                                          !         variable string
                                          !    .FALSE. otherwise
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5tis_variable_str_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Tlock

Signature:

herr_t `H5Tlock`(*hid_t* `dtype_id`)

Purpose:

Locks a datatype.

Description:

`H5Tlock` locks the datatype specified by the `dtype_id` identifier, making it read-only and non-destructible. This is normally done by the library for predefined datatypes so the application does not inadvertently change or delete a predefined type. Once a datatype is locked it can never be unlocked.

Parameters:

hid_t `dtype_id`	Identifier of datatype to lock.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

Name: H5Topen

Signature:

hid_t `H5Topen`(*hid_t* `loc_id`, *const char* * `name`)

hid_t `H5Topen`(*hid_t* `loc_id`, *const char* * `name`, *hid_t* `tapl_id`)

Purpose:

Opens a named datatype.

Description:

`H5Topen` is a macro that is mapped to either `H5Topen1` or `H5Topen2`, depending on the needs of the application.

Such macros are provided to facilitate application compatibility. Their use and mappings are fully described in “API Compatibility Macros in HDF5”; we urge you to read that document closely.

When both the HDF5 Library and the application are built and installed with no specific compatibility flags, `H5Topen` is mapped to the most recent version of the function, currently `H5Topen2`. If the library and/or application is compiled for Release 1.6 emulation, `H5Topen` will be mapped to `H5Topen1`. Function-specific flags are available to override these settings on a function-by-function basis when the application is compiled.

Specific compile-time compatibility flags and the resulting mappings are as follows:

Compatibility setting	`H5Topen` mapping
Global settings	
No compatibility flag	`H5Topen2`
Enable deprecated symbols	`H5Topen2`
Disable deprecated symbols	`H5Topen2`
Emulate Release 1.6 interface	`H5Topen1`
Function-level macros	
`H5Topen_vers = 2`	`H5Topen2`
`H5Topen_vers = 1`	`H5Topen1`

Fortran90 Interface: h5topen_f

```
SUBROUTINE h5topen_f(loc_id, name, type_id, hdferr, tapl_id)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id   ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name   ! Datatype name within file or group
  INTEGER(HID_T), INTENT(OUT) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
  INTEGER(HID_T), OPTIONAL, INTENT(IN) :: tapl_id
                                         ! Datatype access property list id
END SUBROUTINE h5topen_f
```

History:

Release	**C**
1.8.0	The function `H5Topen` renamed to `H5Topen1` and deprecated in this release. The macro `H5Topen` and the function `H5Topen2` introduced in this release.

Name: H5Topen1

Signature:

hid_t `H5Topen1`(*hid_t* `loc_id`, *const char* * `name`)

Purpose:

Opens a named datatype.

Notice:

This function is deprecated in favor of the function `H5Topen2`.

Description:

`H5Topen1` opens a named datatype at the location specified by `loc_id` and returns an identifier for the datatype. `loc_id` is either a file or group identifier. The identifier should eventually be closed by calling `H5Tclose` to release resources.

Parameters:

hid_t `loc_id`	IN: A file or group identifier.
const char * `name`	IN: A datatype name, defined within the file or group identified by `loc_id`.

Returns:

Returns a named datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Topen`.

History:

Release	**C**
1.8.0	The function `H5Topen` renamed to `H5Topen1` and deprecated in this release.

Name: H5Topen2

Signature:

hid_t `H5Topen2`(*hid_t* `loc_id`, *const char* * `name`, *hid_t* `tapl_id`)

Purpose:

Opens a named datatype.

Description:

`H5Topen2` opens a named datatype at the location specified by `loc_id` and returns an identifier for the datatype. `loc_id` is either a file or group identifier. The identifier should eventually be closed by calling `H5Tclose` to release resources.

The named datatype is opened with the dataype access property list `tapl_id`.

Parameters:

hid_t `loc_id`	IN: A file or group identifier.
const char * `name`	IN: A datatype name, defined within the file or group identified by `loc_id`.
hid_t `tapl_id`	IN: Datatype access property list identifier.

Returns:

Returns a named datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface: See listing under `H5Topen`.

History:

Release	**C**
1.8.0	Function introduced in this release.

Name: H5Tpack

Signature:

herr_t `H5Tpack`(*hid_t* `dtype_id`)

Purpose:

Recursively removes padding from within a compound datatype.

Description:

`H5Tpack` recursively removes padding from within a compound datatype to make it more efficient (space-wise) to store that data.

Parameters:

hid_t `dtype_id`	Identifier of datatype to modify.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tpack_f

```
SUBROUTINE h5tpack_f(type_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tpack_f
```

Name: H5Tregister
Signature:
herr_t `H5Tregister`(*H5T_pers_t* `pers`, *const char* * `name`, *hid_t* `src_id`, *hid_t* `dst_id`, *H5T_conv_t* `func`)
Purpose:
Registers a conversion function.
Description:
`H5Tregister` registers a hard or soft conversion function for a datatype conversion path.

The parameter `pers` indicates whether a conversion function is *hard* (`H5T_PERS_HARD`) or *soft* (`H5T_PERS_SOFT`).

A conversion path can have only one hard function. When `pers` is `H5T_PERS_HARD`, `func` replaces any previous hard function. If `pers` is `H5T_PERS_HARD` and `func` is the null pointer, then any hard function registered for this path is removed.

When `pers` is `H5T_PERS_SOFT`, `H5Tregister` adds the function to the end of the master soft list and replaces the soft function in all applicable existing conversion paths. Soft functions are used when determining which conversion function is appropriate for this path.

The `name` is used only for debugging and should be a short identifier for the function.

The path is specified by the source and destination datatypes `src_id` and `dst_id`. For soft conversion functions, only the class of these types is important.

The type of the conversion function pointer is declared as:

```
typedef herr_t (*H5T_conv_t) (hid_t src_id,
                              hid_t dst_id,
                              H5T_cdata_t *cdata,
                              size_t nelmts,
                              size_t buf_stride,
                              size_t bkg_stride,
                              void *buf,
                              void *bkg,
                              hid_t dset_xfer_plist)
```

The `H5T_cdata_t` struct is declared as:

```
typedef struct *H5T_cdata_t (H5T_cmd_t command,
                             H5T_bkg_t need_bkg,
                             hbool_t *recalc,
                             void *priv)
```

The `H5T_conv_t` parameters and the elements of the `H5T_cdata_t` struct are described more fully in the “Data Conversion” section of “The Datatype Interface (H5T)” in the *HDF5 User's Guide*.
Parameters:

H5T_pers_t `pers`	`H5T_PERS_HARD` for hard conversion functions; `H5T_PERS_SOFT` for soft conversion functions.
const char * `name`	Name displayed in diagnostic output.

hid_t `src_id`	Identifier of source datatype.
hid_t `dst_id`	Identifier of destination datatype.
H5T_conv_t `func`	Function to convert between source and destination datatypes.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.3	The following change occurred in the `H5Tconv_t` function: `nelmts` parameter type changed to *size_t*.

Name: H5Tset_cset

Signature:

herr_t `H5Tset_cset(` *hid_t* `dtype_id`, *H5T_cset_t* `cset`)

Purpose:

Sets character set to be used.

Description:

`H5Tset_cset` sets the character set to be used.

HDF5 is able to distinguish between character sets of different nationalities and to convert between them to the extent possible. Valid character set types are:

`H5T_CSET_ASCII (0)`	Character set is US ASCII.

Parameters:

hid_t `dtype_id`	Identifier of datatype to modify.
H5T_cset_t `cset`	Character set type.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_cset_f

```
SUBROUTINE h5tset_cset_f(type_id, cset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id
                                 ! Datatype identifier
  INTEGER, INTENT(IN) :: cset     ! Character set type of a string datatype
                                 ! Possible values of padding type are:
                                 !    H5T_CSET_ASCII_F = 0
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
END SUBROUTINE h5tset_cset_f
```

Name: H5Tset_ebias

Signature:

herr_t `H5Tset_ebias`(*hid_t* `dtype_id`, *size_t* `ebias`)

Purpose:

Sets the exponent bias of a floating-point type.

Description:

`H5Tset_ebias` sets the exponent bias of a floating-point type.

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
size_t `ebias`	Exponent bias value.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_ebias_f

```
SUBROUTINE h5tset_ebias_f(type_id, ebias, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER(SIZE_T), INTENT(IN) :: ebias  ! Datatype exponent bias
                                        ! of a floating-point type,
                                        ! which cannot be 0
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tset_ebias_f
```

Name: H5Tset_fields

Signature:

herr_t `H5Tset_fields`(*hid_t* `dtype_id`, *size_t* `spos`, *size_t* `epos`, *size_t* `esize`, *size_t* `mpos`, *size_t* `msize`)

Purpose:

Sets locations and sizes of floating point bit fields.

Description:

`H5Tset_fields` sets the locations and sizes of the various floating-point bit fields. The field positions are bit positions in the significant region of the datatype. Bits are numbered with the least significant bit number zero.

Fields are not allowed to extend beyond the number of bits of precision, nor are they allowed to overlap with one another.

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
size_t `spos`	Sign position, i.e., the bit offset of the floating-point sign bit.
size_t `epos`	Exponent bit position.
size_t `esize`	Size of exponent in bits.
size_t `mpos`	Mantissa bit position.
size_t `msize`	Size of mantissa in bits.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_fields_f

```
SUBROUTINE h5tset_fields_f(type_id, spos, epos, esize, mpos, msize, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER(SIZE_T), INTENT(IN) :: spos   ! sign bit-position
  INTEGER(SIZE_T), INTENT(IN) :: epos   ! exponent bit-position
  INTEGER(SIZE_T), INTENT(IN) :: esize  ! size of exponent in bits
  INTEGER(SIZE_T), INTENT(IN) :: mpos   ! mantissa bit-position
  INTEGER(SIZE_T), INTENT(IN) :: msize  ! size of mantissa in bits
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tset_fields_f
```

Name: H5Tset_inpad

Signature:

herr_t `H5Tset_inpad`(*hid_t* `dtype_id`, *H5T_pad_t* `inpad`)

Purpose:

Fills unused internal floating point bits.

Description:

If any internal bits of a floating point type are unused (that is, those significant bits which are not part of the sign, exponent, or mantissa), then `H5Tset_inpad` will be filled according to the value of the padding value property `inpad`. Valid padding types are:

H5T_PAD_ZERO (`0`)
: Set background to zeros.

H5T_PAD_ONE (`1`)
: Set background to ones.

H5T_PAD_BACKGROUND (`2`)
: Leave background alone.

Parameters:

hid_t `dtype_id`	Identifier of datatype to modify.
H5T_pad_t `pad`	Padding type.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_inpad_f

```
SUBROUTINE h5tset_inpad_f(type_id, padtype, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id
                                  ! Datatype identifier
  INTEGER, INTENT(IN) :: padtype  ! Padding type for unused bits
                                  ! in floating-point datatypes.
                                  ! Possible values of padding type are:
                                  !    H5T_PAD_ZERO_F = 0
                                  !    H5T_PAD_ONE_F = 1
                                  !    H5T_PAD_BACKGROUND_F = 2
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
END SUBROUTINE h5tset_inpad_f
```

Name: H5Tset_norm

Signature:

herr_t `H5Tset_norm`(*hid_t* `dtype_id`, *H5T_norm_t* `norm`)

Purpose:

Sets the mantissa normalization of a floating-point datatype.

Description:

`H5Tset_norm` sets the mantissa normalization of a floating-point datatype. Valid normalization types are:

H5T_NORM_IMPLIED (`0`)
: MSB of mantissa is not stored, always 1

H5T_NORM_MSBSET (`1`)
: MSB of mantissa is always 1

H5T_NORM_NONE (`2`)
: Mantissa is not normalized

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
H5T_norm_t `norm`	Mantissa normalization type.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_norm_f

```
SUBROUTINE h5tset_norm_f(type_id, norm, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id
                                  ! Datatype identifier
  INTEGER, INTENT(IN) :: norm     ! Mantissa normalization of a
                                  ! floating-point datatype
                                  ! Valid normalization types are:
                                  !    H5T_NORM_IMPLIED_F(0)
                                  !       MSB of mantissa is not stored,
                                  !       always 1
                                  !    H5T_NORM_MSBSET_F(1)
                                  !       MSB of mantissa is always 1
                                  !    H5T_NORM_NONE_F(2)
                                  !       Mantissa is not normalized
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
END SUBROUTINE h5tset_norm_f
```

Name: H5Tset_offset

Signature:

herr_t `H5Tset_offset(` *hid_t* `dtype_id`, *size_t* `offset`)

Purpose:

Sets the bit offset of the first significant bit.

Description:

`H5Tset_offset` sets the bit offset of the first significant bit. The significant bits of an atomic datum can be offset from the beginning of the memory for that datum by an amount of padding. The `offset' property specifies the number of bits of padding that appear to the "right of" the value. That is, if we have a 32-bit datum with 16-bits of precision having the value 0x1122 then it will be laid out in memory as (from small byte address toward larger byte addresses):

Byte Position	**Big-Endian Offset=0**	**Big-Endian Offset=16**	**Little-Endian Offset=0**	**Little-Endian Offset=16**
0:	[pad]	[0x11]	[0x22]	[pad]
1:	[pad]	[0x22]	[0x11]	[pad]
2:	[0x11]	[pad]	[pad]	[0x22]
3:	[0x22]	[pad]	[pad]	[0x11]

If the offset is incremented then the total size is incremented also if necessary to prevent significant bits of the value from hanging over the edge of the datatype.

The offset of an H5T_STRING cannot be set to anything but zero.

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
size_t `offset`	Offset of first significant bit.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_offset_f

```
SUBROUTINE h5tset_offset_f(type_id, offset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER(SIZE_T), INTENT(IN) :: offset ! Datatype bit offset of
                                        ! the first significant bit
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tset_offset_f
```

Name: H5Tset_order

Signature:

herr_t `H5Tset_order(` *hid_t* `dtype_id`, *H5T_order_t* `order`)

Purpose:

Sets the byte order of an atomic datatype.

Description:

`H5Tset_order` sets the byte order of an atomic datatype.

Byte orders currently supported are:

`H5T_ORDER_LE` (`0`)
: Little-endian byte order

`H5T_ORDER_BE` (`1`)
: Big-endian byte order

`H5T_ORDER_VAX` (`2`)
: VAX mixed byte order (not currently supported)

`H5T_ORDER_NONE` (`3`) is also a valid value for `order`, but it has no effect. It is valid only for fixed-length strings and object and region references and specifies “no particular order.”

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
H5T_order_t `order`	Byte order constant.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_order_f

```
SUBROUTINE h5tset_order_f(type_id, order, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id   ! Datatype identifier
  INTEGER, INTENT(IN) :: order            ! Datatype byte order
                                          ! Possible values are:
                                          !    H5T_ORDER_LE_F
                                          !    H5T_ORDER_BE_F
                                          !    H5T_ORDER_VAX_F
  INTEGER, INTENT(OUT) :: hdferr          ! Error code
                                          ! 0 on success and -1 on failure
END SUBROUTINE h5tset_order_f
```

Name: H5Tset_pad

Signature:

herr_t `H5Tset_pad`(*hid_t* `dtype_id`, *H5T_pad_t* `lsb`, *H5T_pad_t* `msb`)

Purpose:

Sets the least and most-significant bits padding types.

Description:

`H5Tset_pad` sets the least and most-significant bits padding types.

H5T_PAD_ZERO (`0`)
: Set background to zeros.

H5T_PAD_ONE (`1`)
: Set background to ones.

H5T_PAD_BACKGROUND (`2`)
: Leave background alone.

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
H5T_pad_t `lsb`	Padding type for least-significant bits.
H5T_pad_t `msb`	Padding type for most-significant bits.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_pad_f

```
SUBROUTINE h5tset_pad_f(type_id, lsbpad, msbpad, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, INTENT(IN) :: lsbpad         ! Padding type of the
                                        ! least significant bit
  INTEGER, INTENT(IN) :: msbpad         ! Padding type of the
                                        ! most significant bit
                                        ! Possible values of padding
                                        ! type are:
                                        !    H5T_PAD_ZERO_F = 0
                                        !    H5T_PAD_ONE_F = 1
                                        !    H5T_PAD_BACKGROUND_F = 2
                                        !    H5T_PAD_ERROR_F = -1
                                        !    H5T_PAD_NPAD_F = 3
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tset_pad_f
```

Name: H5Tset_precision

Signature:

herr_t `H5Tset_precision`(*hid_t* `dtype_id`, *size_t*`precision`)

Purpose:

Sets the precision of an atomic datatype.

Description:

`H5Tset_precision` sets the precision of an atomic datatype. The precision is the number of significant bits which, unless padding is present, is 8 times larger than the value returned by `H5Tget_size`.

If the precision is increased then the offset is decreased and then the size is increased to insure that significant bits do not "hang over" the edge of the datatype.

Changing the precision of an H5T_STRING automatically changes the size as well. The precision must be a multiple of 8.

When decreasing the precision of a floating point type, set the locations and sizes of the sign, mantissa, and exponent fields first.

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
size_t `precision`	Number of bits of precision for datatype.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_precision_f

```
SUBROUTINE h5tset_precision_f(type_id, precision, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id    ! Datatype identifier
  INTEGER(SIZE_T), INTENT(IN) :: precision ! Datatype precision
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
END SUBROUTINE h5tset_precision_f
```

Name: H5Tset_sign

Signature:

herr_t `H5Tset_sign`(*hid_t* `dtype_id`, *H5T_sign_t* `sign`)

Purpose:

Sets the sign property for an integer type.

Description:

`H5Tset_sign` sets the sign property for an integer type.

H5T_SGN_NONE (`0`)
: Unsigned integer type.

H5T_SGN_2 (`1`)
: Two's complement signed integer type.

Parameters:

hid_t `dtype_id`	Identifier of datatype to set.
H5T_sign_t `sign`	Sign type.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_sign_f

```
SUBROUTINE h5tset_sign_f(type_id, sign, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id
                                  ! Datatype identifier
  INTEGER, INTENT(IN) :: sign     ! Sign type for an integer type
                                  ! Possible values are:
                                  !    Unsigned integer type
                                  !       H5T_SGN_NONE_F = 0
                                  !    Two's complement signed integer type
                                  !       H5T_SGN_2_F = 1
                                  !    or error value
                                  !       H5T_SGN_ERROR_F=-1
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
END SUBROUTINE h5tset_sign_f
```

Name: H5Tset_size

Signature:

herr_t `H5Tset_size`(*hid_t* `dtype_id`, *size_t*`size`)

Purpose:

Sets the total size for an atomic datatype.

Description:

`H5Tset_size` sets the total size in bytes, `size`, for a datatype. If the datatype is atomic and size is decreased so that the significant bits of the datatype extend beyond the edge of the new size, then the `offset' property is decreased toward zero. If the `offset' becomes zero and the significant bits of the datatype still hang over the edge of the new size, then the number of significant bits is decreased. The size set for a string should include space for the null-terminator character, otherwise it will not be stored on (or retrieved from) disk. Adjusting the size of a string automatically sets the precision to 8*`size`. A compound datatype may increase or decrease in size as long as its member field is not trailed. All datatypes must have a positive size.

Parameters:

hid_t `dtype_id`	Identifier of datatype to change size.
size_t `size`	Size in bytes to modify datatype.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_size_f

```
SUBROUTINE h5tset_size_f(type_id, size, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  INTEGER(SIZE_T), INTENT(IN) :: size    ! Datatype size
  INTEGER, INTENT(OUT) :: hdferr         ! Error code
                                         ! 0 on success and -1 on failure
END SUBROUTINE h5tset_size_f
```

Name: H5Tset_strpad

Signature:

herr_t `H5Tset_strpad`(*hid_t* `dtype_id`, *H5T_str_t* `strpad`)

Purpose:

Defines the storage mechanism for character strings.

Description:

`H5Tset_strpad` defines the storage mechanism for the string.

The method used to store character strings differs with the programming language:

- ◊ C usually null terminates strings while
- ◊ Fortran left-justifies and space-pads strings.

Valid string padding values, as passed in the parameter `strpad`, are as follows:

H5T_STR_NULLTERM (`0`)
: Null terminate (as C does)

H5T_STR_NULLPAD (`1`)
: Pad with zeros

H5T_STR_SPACEPAD (`2`)
: Pad with spaces (as FORTRAN does)

When converting from a longer string to a shorter string, the behavior is as follows. If the short string is `H5T_STR_NULLPAD` or `H5T_STR_SPACEPAD`, then the string is simply truncated. If the short string is `H5T_STR_NULLTERM`, it is truncated and a null terminator is appended.

When converting from a shorter string to a longer string, the long string is padded on the end by appending nulls or spaces.

Parameters:

hid_t `dtype_id`	Identifier of datatype to modify.
H5T_str_t `strpad`	String padding type.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_strpad_f

```
SUBROUTINE h5tset_strpad_f(type_id, strpad, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id
                                ! Datatype identifier
  INTEGER, INTENT(IN) :: strpad  ! String padding method for a string datatype
                                ! Possible values of padding type are:
                                !    Pad with zeros (as C does):
                                !       H5T_STR_NULLPAD_F(0)
                                !    Pad with spaces (as FORTRAN does):
                                !       H5T_STR_SPACEPAD_F(1)
  INTEGER, INTENT(OUT) :: hdferr ! Error code
END SUBROUTINE h5tset_strpad_f
```

Name: H5Tset_tag

Signature:

herr_t `H5Tset_tag`(*hid_t* `dtype_id` *const char* `*tag`)

Purpose:

Tags an opaque datatype.

Description:

`H5Tset_tag` tags an opaque datatype `dtype_id` with a descriptive ASCII identifier, `tag`.

`tag` is intended to provide a concise description; the maximum size is hard-coded in the HDF5 Library as 256 bytes (`H5T_OPAQUE_TAG_MAX`).

Parameters:

hid_t `dtype_id`	IN: Datatype identifier for the opaque datatype to be tagged.
const char `*tag`	IN: Descriptive ASCII string with which the opaque datatype is to be tagged.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5tset_tag_f

```
SUBROUTINE h5tset_tag_f(type_id, tag, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  CHARACTER(LEN=*), INTENT(IN) :: tag   ! Unique ASCII string with which the
                                        ! opaque datatype is to be tagged
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tset_tag_f
```

History:

Release	**C**
1.6.5	The `H5T_OPAQUE_TAG_MAX` macro constant, specifying the maximum size of an opaque datatype tag, was added in `H5Tpublic.h`.

Name: H5Tunregister

Signature:

herr_t `H5Tunregister(` *H5T_pers_t* `pers`, *const char* `*name`, *hid_t* `src_id`, *hid_t* `dst_id`, *H5T_conv_t* `func`)

Purpose:

Removes a conversion function.

Description:

H5Tunregister removes a conversion function matching certain criteria, like soft or hard conversion, source and destination types, and the conversion function.

If a user is trying to remove a conversion function he registered, all parameters can be used. If he is trying to remove a library s default conversion function, there is no guarantee the `name` and `func` parameters will match user s chosen values. Passing in some values may cause this function to fail. A good practice is to pass in NULL as their values.

All parameters are optional. The missing parameters will be used to generalize the searching criteria.

The conversion function pointer type declaration is described in H5Tregister.

Parameters:

H5T_pers_t `pers`	`H5T_PERS_HARD` for hard conversion functions; `H5T_PERS_SOFT` for soft conversion functions.
const char `*name`	Name displayed in diagnostic output.
hid_t `src_id`	Identifier of source datatype.
hid_t `dst_id`	Identifier of destination datatype.
H5T_conv_t `func`	Function to convert between source and destination datatypes.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.3	The following change occurred in the `H5Tconv_t` function: `nelmts` parameter type changed to *size_t*.

Name: H5Tvlen_create

Signature:

hid_t `H5Tvlen_create(` *hid_t* `base_type_id`)

Purpose:

Creates a new variable-length datatype.

Description:

`H5Tvlen_create` creates a new variable-length (VL) datatype.

The base datatype will be the datatype that the sequence is composed of, characters for character strings, vertex coordinates for polygon lists, etc. The base type specified for the VL datatype can be of any HDF5 datatype, including another VL datatype, a compound datatype or an atomic datatype.

When necessary, use `H5Tget_super` to determine the base type of the VL datatype.

The datatype identifier returned from this function should be released with `H5Tclose` or resource leaks will result.

Parameters:

hid_t `base_type_id`	Base type of datatype to create.

See Also:

H5Dget_vlen_buf_size
H5Dvlen_reclaim

Returns:

Returns datatype identifier if successful; otherwise returns a negative value.

Fortran90 Interface: h5tvlen_create_f

```
SUBROUTINE h5tvlen_create_f(type_id, vltype_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id    ! Datatype identifier of base type
                                           ! Base type can only be atomic
  INTEGER(HID_T), INTENT(OUT) :: vltype_id ! VL datatype identifier
  INTEGER, INTENT(OUT) :: hdferr           ! Error code
END SUBROUTINE h5tvlen_create_f
```

History:

Release	**Fortran90**
1.4.5	Function introduced in this release.

H5Z: Filter and Compression Interface

Filter and Compression API Functions

These functions enable the user to configure new filters for the local environment.

- H5Zfilter_avail
- H5Zget_filter_info
- H5Zregister
- H5Zunregister

The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

- h5zfilter_avail_f
- h5zget_filter_info_f
- h5zunregister_f

HDF5 supports a filter pipeline that provides the capability for standard and customized raw data processing during I/O operations. HDF5 is distributed with a small set of standard filters such as compression (gzip, SZIP, and a shuffling algorithm) and error checking (Fletcher32 checksum). For further flexibility, the library allows a user application to extend the pipeline through the creation and registration of customized filters.

The flexibility of the filter pipeline implementation enables the definition of additional filters by a user application. A filter

- is associated with a dataset when the dataset is created,
- can be used only with chunked data
  (i.e., datasets stored in the `H5D_CHUNKED` storage layout), and
- is applied independently to each chunk of the dataset.

The HDF5 library does not support filters for contiguous datasets because of the difficulty of implementing random access for partial I/O. Compact dataset filters are not supported because it would not produce significant results.

Filter identifiers for the filters distributed with the HDF5 Library are as follows:

`H5Z_FILTER_DEFLATE`	The gzip compression, or deflation, filter
`H5Z_FILTER_SZIP`	The SZIP compression filter
`H5Z_FILTER_NBIT`	The N-bit compression filter
`H5Z_FILTER_SCALEOFFSET`	The scale-offset compression filter
`H5Z_FILTER_SHUFFLE`	The shuffle algorithm filter
`H5Z_FILTER_FLETCHER32`	The Fletcher32 checksum, or error checking, filter

Custom filters that have been registered with the library will have additional unique identifiers.

See *The Dataset Interface (H5D)* in the *HDF5 User's Guide* for further information regarding data compression.

Name: H5Zfilter_avail

Signature:

htri_t `H5Zfilter_avail`(*H5Z_filter_t* `filter`)

Purpose:

Determines whether a filter is available.

Description:

`H5Zfilter_avail` determines whether the filter specified in `filter` is available to the application.

Parameters:

H5Z_filter_t `filter`	IN: Filter identifier. See the introduction to this section of the reference manual for a list of valid filter identifiers.

Returns:

Returns a Boolean value (`TRUE`/`FALSE`) if successful; otherwise returns a negative value.

Fortran90 Interface: h5zfilter_avail_f

```
SUBROUTINE h5zfilter_avail_f(filter, status, hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(IN)  :: filter     ! Filter
                                     ! Valid values are:
                                     !    H5Z_FILTER_DEFLATE_F
                                     !    H5Z_FILTER_SHUFFLE_F
                                     !    H5Z_FILTER_FLETCHER32_F
                                     !    H5Z_FILTER_SZIP_F
  LOGICAL, INTENT(OUT) :: status     ! Flag indicating whether
                                     ! filter is available:
                                     !    .TRUE.
                                     !    .FALSE.
END SUBROUTINE h5zfilter_avail_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

Name: H5Zget_filter_info
Signature:

herr_t `H5Zget_filter_info`(*H5Z_filter_t* `filter`, *unsigned int* *`filter_config`)

Purpose:

Retrieves information about a filter.

Description:

`H5Zget_filter_info` retrieves information about a filter. At present, this means that the function retrieves a filter's configuration flags, indicating whether the filter is configured to decode data, to encode data, neither, or both.

If `filter_config` is not set to NULL prior to the function call, the returned parameter contains a bit field specifying the available filter configuration. The configuration flag values can then be determined through a series of bitwise AND operations, as described below.

Valid filter configuration flags include the following:

`H5Z_FILTER_CONFIG_ENCODE_ENABLED`	Encoding is enabled for this filter. In Fortran, `H5Z_FILTER_ENCODE_ENABLED_F`.
`H5Z_FILTER_CONFIG_DECODE_ENABLED`	Decoding is enabled for this filter. In Fortran, `H5Z_FILTER_DECODE_ENABLED_F`.

(These flags are defined for C in the HDF5 Library source code file `H5Zpublic.h`.)

A bitwise AND of the returned `filter_config` and a valid filter configuration flag will reveal whether the related configuration option is available. For example, if the value of

```
H5Z_FILTER_CONFIG_ENCODE_ENABLED & filter_config
```

is true, i.e., greater than `0` (zero), the queried filter is configured to encode data; if the value is `FALSE`, i.e., equal to `0` (zero), the filter is not so configured.

If a filter is not encode-enabled, the corresponding `H5Pset_*` function will return an error if the filter is added to a dataset creation property list (which is required if the filter is to be used to encode that dataset). For example, if the `H5Z_FILTER_CONFIG_ENCODE_ENABLED` flag is not returned for the SZIP filter, `H5Z_FILTER_SZIP`, a call to `H5Pset_szip` will fail.

If a filter is not decode-enabled, the application will not be able to read an existing file encoded with that filter.

This function should be called, and the returned `filter_config` analyzed, before calling any other function, such as `H5Pset_szip`, that might require a particular filter configuration.

Parameters:

H5Z_filter_t `filter`

IN: Identifier of the filter to query. See the introduction to this section of the reference manual for a list of valid filter identifiers.

unsigned int *`filter_config`

OUT: A bit field encoding the returned filter information

Returns:

Returns a non-negative value on success, a negative value on failure.

Fortran90 Interface:

```
SUBROUTINE h5zget_filter_info_f(filter, config_flags, hdferr)

  IMPLICIT NONE
  INTEGER, INTENT(IN)  :: filter        ! Filter, may be one of the
                                        ! following:
                                        !     H5Z_FILTER_DEFLATE_F
                                        !     H5Z_FILTER_SHUFFLE_F
                                        !     H5Z_FILTER_FLETCHER32_F
                                        !     H5Z_FILTER_SZIP_F
  INTEGER, INTENT(OUT) :: config_flags  ! Bit field indicating whether
                                        ! a filter's encoder and/or
                                        ! decoder are available
  INTEGER, INTENT(OUT) :: hdferr        ! Error code

END SUBROUTINE h5zfilter_avail_f
```

History:

Release	**C**
1.6.3	Function introduced in this release. Fortran subroutine introduced in this release.

Name: H5Zregister
Signature:
herr_t `H5Zregister`(*const H5Z_class_t* *`filter_class`))
Purpose:
Registers new filter.
Description:
`H5Zregister` registers a new filter with the HDF5 library.

Making a new filter available to an application is a two-step process. The first step is to write the three filter callback functions described below: `can_apply`, `set_local`, and `filter`. This call to `H5Zregister`, registering the filter with the library, is the second step. The `can_apply` and `set_local` fields can be set to `NULL` if they are not required for the filter being registered.

`H5Zregister` accepts a single parameter, a pointer to a buffer for the `filter_class` data structure. That data structure must conform to one of the following definitions:

```
typedef struct H5Z_class1_t {
    H5Z_filter_t id;
    const char  *name;
    H5Z_can_apply_func_t can_apply;
    H5Z_set_local_func_t set_local;
    H5Z_func_t filter;
} H5Z_class1_t;

typedef struct H5Z_class2_t {
    int version;
    H5Z_filter_t id;
    unsigned encoder_present;
    unsigned decoder_present;
    const char  *name;
    H5Z_can_apply_func_t can_apply;
    H5Z_set_local_func_t set_local;
    H5Z_func_t filter;
} H5Z_class2_t;
```

`version` is a libray-defined value reporting the version number of the *H5Z_class_t* struct. This currently must be set to H5Z_CLASS_T_VERS.

`id` is the identifier for the new filter. This is a user-defined value between `H5Z_FILTER_RESERVED` and `H5Z_FILTER_MAX`. These values are defined in the HDF5 source file `H5Zpublic.h`, but the symbols `H5Z_FILTER_RESERVED` and `H5Z_FILTER_MAX` should always be used instead of the literal values.

`encoder_present` is a library-defined value indicating whether the filter’s encoding capability is available to the application.

`decoder_present` is a library-defined value indicating whether the filter’s encoding capability is available to the application.

`name` is a descriptive comment used for debugging, may contain a descriptive name for the filter, and may be the null pointer.

`can_apply`, described in detail below, is a user-defined callback function which determines whether the combination of the dataset creation property list values, the datatype, and the dataspace represent a valid combination to apply this filter to.

`set_local`, described in detail below, is a user-defined callback function which sets any parameters that are specific to this dataset, based on the combination of the dataset creation property list values, the datatype, and the dataspace.

`filter`, described in detail below, is a user-defined callback function which performs the action of the filter.

The statistics associated with a filter are not reset by this function; they accumulate over the life of the library.

`H5Z_class_t` is a macro which maps to either `H5Z_class1_t` or `H5Z_class2_t`, depending on the needs of the application. To affect only this macro, `H5Z_class_t_vers` may be defined to either `1` or `2`. Otherwise, it will behave in the same manner as other API compatibility macros. See “API Compatibility Macros in HDF5” for more information. `H5Z_class1_t` matches the `H5Z_class_t` structure that is used in the 1.6.x versions of the HDF5 library.

H5Zregister will automatically detect which structure type has been passed in, regardless of the mapping of the `H5Z_class_t` macro. However, the application must make sure that the fields are filled in according to the correct structure definition if the macro is used to declare the structure.

The callback functions
Before `H5Zregister` can link a filter into an application, three callback functions must be defined as described in the HDF5 Library header file `H5Zpublic.h`.

The *can apply* callback function is defined as follows:

typedef herr_t (*`H5Z_can_apply_func_t`) (*hid_t* `dcpl_id`, *hid_t* `type_id`, *hid_t* `space_id`)

Before a dataset is created, the *can apply* callbacks for any filters used in the dataset creation property list are called with the dataset's dataset creation property list, `dcpl_id`, the dataset's datatype, `type_id`, and a dataspace describing a chunk, `space_id`, (for chunked dataset storage).

This callback must determine whether the combination of the dataset creation property list settings, the datatype, and the dataspace represent a valid combination to which to apply this filter. For example, an invalid combination may involve the filter not operating correctly on certain datatypes, on certain datatype sizes, or on certain sizes of the chunk dataspace.

This callback can be the `NULL` pointer, in which case the library will assume that the filter can be applied to a dataset with any combination of dataset creation property list values, datatypes, and dataspaces.

The *can apply* callback function must return a positive value for a valid combination, zero for an invalid combination, and a negative value for an error.

The *set local* callback function is defined as follows:

typedef herr_t (*`H5Z_set_local_func_t`) (*hid_t* `dcpl_id`, *hid_t* `type_id`, *hid_t* `space_id`)

After the *can apply* callbacks are checked for a new dataset, the *set local* callback functions for any filters used in the dataset creation property list are called. These callbacks receive `dcpl_id`, the dataset's private copy of the dataset creation property list passed in to `H5Dcreate` (i.e. not the actual property list passed in to `H5Dcreate`); `type_id`, the datatype identifier passed in to `H5Dcreate`, which is not copied and should not be modified; and `space_id`, a dataspace describing the chunk (for chunked dataset storage), which should also not be modified.

The *set local* callback must set any filter parameters that are specific to this dataset, based on the combination of the dataset creation property list values, the datatype, and the dataspace. For example, some filters perform different actions based on different datatypes, datatype sizes, numbers of dimensions, or dataspace sizes.

The *set local* callback may be the `NULL` pointer, in which case, the library will assume that there are no dataset-specific settings for this filter.

The *set local* callback function must return a non-negative value on success and a negative value for an error.

The *filter operation* callback function, defining the filter's operation on the data, is defined as follows:

typedef size_t (*`H5Z_func_t`) (*unsigned int* `flags`, *size_t* `cd_nelmts`, *const unsigned int* `cd_values[]`, *size_t* `nbytes`, *size_t* *`buf_size`, *void* **`buf`)

The parameters `flags`, `cd_nelmts`, and `cd_values` are the same as for the function `H5Pset_filter`. The one exception is that an additional flag, `H5Z_FLAG_REVERSE`, is set when the filter is called as part of the input pipeline.

The parameter `*buf` points to the input buffer which has a size of `*buf_size` bytes, `nbytes` of which are valid data.

The filter should perform the transformation in place if possible. If the transformation cannot be done in place, then the filter should allocate a new buffer with `malloc()` and assign it to `*buf`, assigning the allocated size of that buffer to `*buf_size`. The old buffer should be freed by calling `free()`.

If successful, the *filter operation* callback function returns the number of valid bytes of data contained in `*buf`. In the case of failure, the return value is `0` (zero) and all pointer arguments are left unchanged.

Parameters:

const H5Z_class_t *`filter_class`	IN: A pointer to a buffer for the struct containing filter-definition information.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface:

None.

History:

Release	**C**
1.6.0	This function was substantially revised in Release 1.6.0 with a new `H5Z_class_t` struct and new *set local* and *can apply* callback functions.
1.8.0	The fields `version`, `encoder_present`, and `decoder_present` were added to the `H5Z_class_t` struct in this release.
1.8.3	`H5Z_class_t` renamed to `H5Z_class2_t`, `H5Z_class1_t` structure introduced for backwards compatibility with release 1.6.x, and `H5Z_class_t` macro introduced in this release. Function modified to accept either structure type.

Name: H5Zunregister

Signature:

herr_t `H5Zunregister`(*H5Z_filter_t* `filter`)

Purpose:

Unregisters a filter.

Description:

`H5Zunregister` unregisters the filter specified in `filter`.

After a call to `H5Zunregister`, the filter specified in `filter` will no longer be available to the application.

Parameters:

H5Z_filter_t `filter`	IN: Identifier of the filter to be unregistered. See the introduction to this section of the reference manual for a list of identifiers for standard filters distributed with the HDF5 Library.

Returns:

Returns a non-negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5zunregister_f

```
SUBROUTINE h5zunregister_f(filter, hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(IN)  :: filter  ! Filter; one of the possible values:
                                  !    H5Z_FILTER_DEFLATE_F
                                  !    H5Z_FILTER_SHUFFLE_F
                                  !    H5Z_FILTER_FLETCHER32_F
                                  !    H5Z_FILTER_SZIP_F
  INTEGER, INTENT(OUT) :: hdferr  ! Error code
                                  ! 0 on success, and -1 on failure
END SUBROUTINE h5zunregister_f
```

History:

Release	**C**
1.6.0	Function introduced in this release.

HDF5 Tools

HDF5 Tool Interfaces

HDF5-related tools are available to assist the user in a variety of activities, including examining or managing HDF5 files, converting raw data between HDF5 and other special-purpose formats, moving data and files between the HDF4 and HDF5 formats, measuring HDF5 library performance, and managing HDF5 library and application compilation, installation and configuration. Unless otherwise specified below, these tools are distributed and installed with HDF5.

- User utilities:
    - h5dump -- Enables a user to examine the contents of an HDF5 file and dump those contents to an ASCII file.
    - h5ls -- Lists specified features of HDF5 file contents.
    - h5diff and ph5diff -- Compare two HDF5 files and report the differences.
    - h5repack -- Copies an HDF5 file to a new file with or without compression and/or chunking.
    - h5repart -- Repartitions a file, creating a family of files.
    - h5jam -- Adds a user block to the front of an HDF5 file .
    - h5unjam -- Splits an existing user block from an HDF5 file, placing it in a separate file.
    - h5copy -- Copies HDF5 objects from a file to a new file
    - h5mkgrp -- Creates a new HDF5 group in a file
    - h5stat -- Reports statistics regarding an HDF5 file and the objects in the file.
    - h5perf -- Measures Parallel HDF5 performance.
    - h5perf_serial -- Measures HDF5 serial performance.

- Configuration and library management utilities:
    - h5redeploy -- Updates HDF5 compiler tools after an HDF5 software installation in a new location.
    - h5cc and h5pcc -- Simplify the compilation of HDF5 programs written in C.
    - h5fc and h5pfc -- Simplify the compilation of HDF5 programs written in Fortran90.
    - h5c++ -- Simplifies the compilation of HDF5 programs written in C++.

- Java-based tools for HDF5 for viewing, manipulating, and generating HDF4 and HDF5 files:
  (Distributed separately; external link is `http://www.hdfgroup.org/hdf-java-html/`)
    - `HDFview` -- a browser that works with both HDF4 and HDF5 files and can be used to transfer data between the two formats
    - Java interfaces for both the HDF4 and HDF5 libraries
    - Other HDF4- and HDF5-related products

- Data conversion utilities:
    - h5import -- Imports data into an existing or new HDF5 file.
    - gif2h5 -- Converts a GIF file to an HDF5 file.
    - h52gif -- Converts images in an HDF5 file to a GIF file.

- HDF5/HDF4 conversion tools:
  (Distributed separately; external link is `http://www.hdfgroup.org/h4toh5/`.)
    - H4toH5 Conversion Library -- Provides APIs for use in tools that perform customized conversions of HDF4 files to HDF5 files.

- h5toh4 -- Converts an HDF5 file to an HDF4 file.
- h4toh5 -- Converts an HDF4 file to an HDF5 file.

- Other tools, including third-party and commercial utilities and applications:
  (Distributed separately; external link is `http://www.hdfgroup.org/tools5.html`*.)*

Tool Name: h5dump
Syntax:

`h5dump [`*OPTIONS*`]` *file*

Purpose:

Displays HDF5 file contents.

Description:

`h5dump` enables the user to examine the contents of an HDF5 file and dump those contents, in human readable form, to an ASCII file.

`h5dump` dumps HDF5 file content to standard output. It can display the contents of the entire HDF5 file or selected objects, which can be groups, datasets, a subset of a dataset, links, attributes, or datatypes.

The `--header` option displays object header information only.

Names are the absolute names of the objects. `h5dump` displays objects in the order same as the command order. If a name does not start with a slash, `h5dump` begins searching for the specified object starting at the root group.

If an object is hard linked with multiple names, `h5dump` displays the content of the object in the first occurrence. Only the link information is displayed in later occurrences.

`h5dump` assigns a name for any unnamed datatype in the form of #*oid1*`:`*oid2*, where *oid1* and *oid2* are the object identifiers assigned by the library. The unnamed types are displayed within the root group.

Datatypes are displayed with standard type names. For example, if a dataset is created with `H5T_NATIVE_INT` type and the standard type name for integer on that machine is `H5T_STD_I32BE`, `h5dump` displays `H5T_STD_I32BE` as the type of the dataset.

`h5dump` can also dump a subset of a dataset. This feature operates in much the same way as hyperslabs in HDF5; the parameters specified on the command line are passed to the function `H5Sselect_hyperslab` and the resulting selection is displayed.

The `h5dump` output is described in detail in the *DDL for HDF5*, the *Data Description Language* document.

Note: It is not permissible to specify multiple attributes, datasets, datatypes, groups, or soft links with one flag. For example, one may not issue the command

WRONG: `h5dump -a /attr1 /attr2 foo.h5` to display both `/attr1` and `/attr2`. One must issue the following command:

CORRECT: `h5dump -a /attr1 -a /attr2 foo.h5`

It's possible to select the file driver with which to open the HDF5 file by using the --filedriver (-f) command-line option. Acceptable values for the --filedriver option are: "sec2", "family", "split", and "multi". If the file driver flag isn't specified, then the file will be opened with each driver in turn and in the order specified above until one driver succeeds in opening the file.

One byte integer type data is displayed in decimal by default. When displayed in ASCII, a non-printable code is displayed in 3 octal digits preceeded by a back-slash unless there is a C language escape sequence for it. For example, CR and LF are printed as \r and \n. Though the NUL code is represented as \0 in C, it is printed as \000 to avoid ambiguity as illustrated in the following 1 byte char data (since this is not a string, embedded NUL is possible).

```
141 142 143 000 060 061 062 012
  a   b   c  \0   0   1   2  \n
```

h5dump prints them as "abc\000012\n". But if h5dump prints NUL as \0, the output is "abc\0012\n" which is ambiguous.

XML Output:

With the `--xml` option, `h5dump` generates XML output. This output contains a complete description of the file, marked up in XML. The XML conforms to the HDF5 Document Type Definition (DTD) available at `http://www.hdfgroup.org/DTDs/HDF5-File.dtd`.

The XML output is suitable for use with other tools, including the HDF5 Java Tools.

Options and Parameters:

`-h` or `--help`	Print a usage message and exit.
`-n` or `--contents`	Print a list of the file contents and exit.
`-B` or `--bootblock`	Print the content of the boot block.
`-H` or `--header`	Print the header only; no data is displayed.
`-A` or `--onlyattr`	Print the header and value of attributes; data of datasets is not displayed.
`-i` or `--object-ids`	Print the object ids.
`-r` or `--string`	Print 1-byte integer datasets as ASCII.
`-e` or `--escape`	Escape non-printing characters.
`-V` or `--version`	Print version number and exit.
`-a` *`P`* or `--attribute=`*`P`*	Print the specified attribute.
`-d` *`P`* or `--dataset=`*`P`*	Print the specified dataset.
`-y` or `--noindex`	Do not print array indices with data.
`-p` or `--properties`	Print information regarding dataset properties, including filters, storage layout, fill value, and allocation time. The filter output lists any filters used with a dataset, including the type of filter, its name, and any filter parameters. The storage layout output specifies the dataset layout (chunked, compact, or contiguous), the size in bytes of the dataset on disk, and, if a compression filter is associated with the dataset, the compression ratio. The compression ratio is computed as (uncompressed size)/(compressed size). The fill value output includes the fill value datatype and value. The allocation time output displays the allocation time as specified with `H5Pset_alloc_time`.
`-f` *`D`* or `--filedriver=`*`D`*	Specify which driver to open the file with.
`-g` *`P`* or `--group=`*`P`*	Print the specified group and all members.
`-l` *`P`* or `--soft-link=`*`P`*	Print the value(s) of the specified soft link.
`-o` *`F`* or `--output=`*`F`*	Output raw data into file F.

`-b` *B* or `--binary=`*B*	Output dataset to a binary file using the datatype specified by *B*. *B* must have one of the following values: `LE` Little-endian `BE` Big-endian `MEMORY` Memory datatype `FILE` File datatype Recommended usage is with the `-d` and `-o` options.
`-t` *P* or `--datatype=`*P*	Print the specified named datatype.
`-w` *N* or `--width=`*N*	Set the number of columns of output.
`-m` *T* or `--format=`*T*	Set the floating point output format. *T* is a string defining the floating point format, e.g., `'%.3f'`.
`-q` *Q* or `--sort_by=`*Q*	Sort groups and attributes by the specified index type, *Q*. Valid values of *Q* are as follows: `name` Alpha-numeric index by name *(Default)* `creation_order` Index by creation order
`-z` *Z* or `--sort_order=`*Z*	Sort groups and attributes in the specified order, *Z*. Valid values of *Z* are as follows: `ascending` Sort in ascending order *(Default)* `descending` Sort in descending order
`-x` or `--xml`	Output XML using XML schema (default) instead of DDL.
`-u` or `--use-dtd`	Output XML using XML DTD instead of DDL.
`-D` *U* or `--xml-dtd=`*U*	In XML output, refer to the DTD or schema at *U* instead of the default schema/DTD.
`-X` *S* or `--xml-dns=`*S*	In XML output, (XML Schema) use qualified names in the XML: ":": no namespace, default: "hdf5:"
`-s` *L* or `--start=`*L*	Offset of start of subsetting selection. Default: the beginning of the dataset.
`-S` *L* or `--stride=`*L*	Hyperslab stride. Default: 1 in all dimensions.
`-c` *L* or `--count=`*L*	Number of blocks to include in the selection.

`-k` *`L`* or `--block=`*`L`*	Size of block in hyperslab. Default: 1 in all dimensions.
`--`	Indicates that the following argument is not an option. E.g., to dump a file called `-f', use h5dump -- -f. *(This option is necessary only when the name of the file to be examined starts with a dash (`-`), which could confuse the tool's command-line parser.)*
file	The file to be examined.

The option parameters listed above are defined as follows:

D	which file driver to use in opening the file. Acceptable values are "sec2", "family", "split", and "multi". Without the file driver flag, the file will be opened with each driver in turn and in the order specified above until one driver succeeds in opening the file.
P	The full path from the root group to the object
F	A filename
N	An integer greater than 1
L	A list of integers, the number of which is equal to the number of dimensions in the dataspace being queried
U	A URI (as defined in [IETF RFC 2396], updated by [IETF RFC 2732]) that refers to the DTD to be used to validate the XML
B	The form of binary output:

`MEMORY`	for a memory type
`FILE`	for the file type
`LE` or `BE`	for pre-existing little- or big-endian types

Subsetting parameters can also be expressed in a convenient compact form, as follows:

```
--dataset="/foo/mydataset[START;STRIDE;COUNT;BLOCK]"
```

All of the semicolons (`;`) are required, even when a parameter value is not specified. When not specified, default parameter values are used.

Examples:

1. Dump the group `/GroupFoo/GroupBar` in the file `quux.h5`:
```
h5dump -g /GroupFoo/GroupBar quux.h5
```

2. Dump the dataset `Fnord`, which is in the group `/GroupFoo/GroupBar` in the file `quux.h5`:
```
h5dump -d /GroupFoo/GroupBar/Fnord quux.h5
```

3. Dump the attribute `metadata` of the dataset `Fnord`, which is in the group `/GroupFoo/GroupBar` in the file `quux.h5`:
```
h5dump -a /GroupFoo/GroupBar/Fnord/metadata quux.h5
```

4. Dump the attribute `metadata` which is an attribute of the root group in the file `quux.h5`:
```
h5dump -a /metadata quux.h5
```

5. Produce an XML listing of the file `bobo.h5`, saving the listing in the file `bobo.h5.xml`:
```
h5dump --xml bobo.h5 > bobo.h5.xml
```

6. Dump a subset of the dataset `/GroupFoo/databar/` in the file `quux.h5`:
```
h5dump -d /GroupFoo/databar --start="1,1" --stride="2,3"
    --count="3,19" --block="1,1" quux.h5
```

7. The same example, using the short form to specify the subsetting parameters:

```
h5dump -d "/GroupFoo/databar[1,1;2,3;3,19;1,1]" quux.h5
```

8. Dump a binary copy of the dataset `/GroupD/FreshData/` in the file `quux.h5`, with data written in little-endian form, to the output file `FreshDataD.bin`:

```
h5dump -d "/GroupD/FreshData" -b LE
    -o "FreshDataD.bin" quux.h5
```

Current Status:

The current version of `h5dump` displays the following information:

◊ Group
- · group attribute (see Attribute)
- · group member

◊ Dataset
- · dataset attribute (see Attribute)
- · dataset type (see Datatype)
- · dataset space (see Dataspace)
- · dataset data

◊ Attribute
- · attribute type (see Datatype)
- · attribute space (see Dataspace)
- · attribute data

◊ Datatype
- · integer type
  - H5T_STD_I8BE, H5T_STD_I8LE, H5T_STD_I16BE, ...
- · floating point type
  - H5T_IEEE_F32BE, H5T_IEEE_F32LE, H5T_IEEE_F64BE, ...
- · string type
- · compound type
  - named, unnamed and transient compound type
  - integer, floating or string type member
- · opaque types
- · reference type
  - object references
  - data regions
- · enum type
- · variable-length datatypes
  - atomic types only
  - scalar or single dimensional array of variable-length types supported

◊ Dataspace
- · scalar and simple space

◊ Soft link

◊ Hard link

◊ Loop detection

See Also:

- ♦ HDF5 Data Description Language syntax at *DDL for HDF5*
- ♦ HDF5 XML Schema at http://www.hdfgroup.org/DTDs/HDF5-File.xsd
- ♦ HDF5 XML information at http://www.hdfgroup.org/HDF5/XML/

History:

Release	**Command Line Tool**

1.6.5	The following options added in this release:
	`-n` or `--contents`
	`-e` or `--escape`
	`-y` or `--noindex`
	`-p` or `--properties`
	`-b` or `--binary`

1.8.0	The following options added in this release:
	`-q` or `--sort_by`
	`-z` or `--sort_order`

1.8.1	Compression ratio added to -p option in this release.

Tool Name: h5ls

Syntax:

`h5ls [`*OPTIONS*`]` *file* `[`*OBJECTS...*`]`

Purpose:

Prints information about a file or dataset.

Description:

`h5ls` prints selected information about file objects in the specified format.

Options and Parameters:

`-h` or `-?` or `--help`	Print a usage message and exit.
`-a` or `--address`	Print addresses for raw data.
`-d` or `--data`	Print the values of datasets.
`-e` or `--errors`	Show all HDF5 error reporting.
`-f` or `--full`	Print full path names instead of base names.
`-g` or `--group`	Show information about a group, not its contents.
`-l` or `--label`	Label members of compound datasets.
`-r` or `--recursive`	List all groups recursively, avoiding cycles.
`-s` or `--string`	Print 1-bytes integer datasets as ASCII.
`-S` or `--simple`	Use a machine-readable output format.
`-w`*N* or `--width=`*N*	Set the number of columns of output.
`-v` or `--verbose`	Generate more verbose output.
`-V` or `--version`	Print version number and exit.
`-x` or `--hexdump`	Show raw data in hexadecimal format.
file	The file name may include a printf(3C) integer format such as `%%05d` to open a file family.
objects	Each object consists of an HDF5 file name optionally followed by a slash and an object name within the file (if no object is specified within the file then the contents of the root group are displayed). The file name may include a `printf(3C)` integer format such as "%05d" to open a file family.

Last modified: 12 May 2009

Tool Name: h5diff

Syntax:

`h5diff` [*OPTIONS*] *file1 file2* [*object1* [*object2*]]
`ph5diff` [*OPTIONS*] *file1 file2* [*object1* [*object2*]]

Purpose:

Compare two HDF5 files and report the differences.

Description:

`h5diff` and `ph5diff` are command line tools that compare two HDF5 files, *file1* and *file2*, and report the differences between them. `h5diff` is for serial use while `ph5diff` is for use in parallel environments.

Optionally, `h5diff` and `ph5diff` will compare two objects within these files. If only one object, *object1*, is specified, `h5diff` will compare *object1* in *file1* with *object1* in *file2*. If two objects, *object1* and *object2*, are specified, `h5diff` will compare *object1* in *file1* with *object2* in *file2*.

object1 and *object2* can be groups, datasets, named datatypes, or links and must be expressed as absolute paths from the respective file's root group.

◊ If these objects are groups, the name string is compared.
◊ If these objects are datasets, array rank and dimensions, datatypes, and data values are compared.
◊ If these objects are named datatypes, the comparison is based on the return value of `H5Tequal`.
◊ If these objects are links, the link value is compared.
(The link value is the string that is often referred to as the name of the linked object.)

`h5diff` and `ph5diff` have the following output modes:

Normal mode	Prints the number of differences found and where they occurred.
Report mode (`-r`)	Prints the above plus the differences.
Verbose mode (`-v`)	Prints all of the above plus a list of objects and warnings.
Quiet mode (`-q`)	Prints no output. (`h5diff` always returns an exit code of `1` when differences are found.)

`h5diff` and NaNs:

`h5diff` detects when a value in a dataset is a NaN (a "not a number" value), but does not differentiate among various types of NaNs. Thus, when one NaN is compared with another NaN, `h5diff` treats them as equal; when a NaN is compared with a valid number, `h5diff` treats them as not equal.

Note that NaN detection is computationally expensive and slows `h5diff` performance dramatically. If you do not have NaNs in your files, or do not care about NaNs, use the `-N` option below to turn off NaN detection. Similarly, if `h5diff -N` produces unexpected differences, running `h5diff` without `-N` should reveal whether any of the differences are associated with NaN values.

Difference between `h5diff` and `ph5diff`:

With the following exception, `h5diff` and `ph5diff` behave identically. With `ph5diff`, the comparison of objects is shared across multiple processors, with the comparison of each pair of objects assigned to a single processor. I.e., the comparison of a single object, even a very large dataset, in each file is not shared.

Options and Parameters:

`-h` or `--help`	Print help message.
`-V` or `--version`	Print version number and exit.
`-r` or `--report`	Report mode — Print the differences.
`-v` or `--verbose`	Verbose mode — Print the differences, a list of objects, and warnings.
`-q` or `--quiet`	Quiet mode — Do not print output.
`-N` or `--nan`	Disables NaN detection; see "`h5diff` and NaNs" above.
`-n` *count* or `--count=`*count*	Print difference up to *count* differences, then stop. *count* must be a positive integer.
`-d` *delta* or `--delta=`*delta*	Print only differences that are greater than the limit *delta*. *delta* must be a positive number. The comparison criterion is whether the absolute value of the difference of two corresponding values is greater than *delta* (e.g., `|a–b| >` ***`delta`***, where `a` is a value in *file1* and `b` is a value in *file2*).
`-p` *relative* or `--relative=`*relative*	Print only differences that are greater than a relative error. *relative* must be a positive number. The comparison criterion is whether the absolute value of the difference `1` and the ratio of two corresponding values is greater than *relative* (e.g., `|1–(b/a)| >` ***`relative`*** where `a` is a value in *file1* and `b` is a value in *file2*).
file1 *file2*	The HDF5 files to be compared.
object1 *object2*	Specific object(s) within the files to be compared.

Returns:

Returns `1` (one) if differences are found, `0` (zero) if no differences are found, and `2` (two) if the call fails.

Examples:

The following `h5diff` call compares the object `/a/b` in `file1` with the object `/a/c` in `file2`:

```
h5diff file1 file2 /a/b /a/c
```

This `h5diff` call compares the object `/a/b` in `file1` with the same object in `file2`:

```
h5diff file1 file2 /a/b
```

And this `h5diff` call compares all objects in both files:

```
h5diff file1 file2
```

History:

Release	**Change**
1.6.0	`h5diff` introduced in this release.
1.8.0	`ph5diff` introduced in this release. `h5diff` command line syntax changed in this release.
1.8.2 and 1.6.8	Return value on failure changed in this release.

Tool Name: h5repack
Syntax:

`h5repack` [*OPTIONS*] *in_file out_file*
`h5repack -i` *in_file* `-o` *out_file* [*OPTIONS*]

Purpose:

Copies an HDF5 file to a new file with or without compression and/or chunking.

Description:

`h5repack` is a command line tool that applies HDF5 filters to an input file *in_file*, saving the output in a new output file, *out_file*.

Options and Parameters:

`-i` *in_file*
Input HDF5 file

`-o` *out_file*
Output HDF5 file

`-h` or `--help`
Print help message.

`-v` or `--verbose`
Print verbose output.

`-V` or `--version`
Print version number.

`-n` or `--native`
Use native HDF5 datatypes when repacking.
(Default behavior is to use original file datatypes.)
Note that this is a change in default behavior; prior to Release 1.6.6, `h5repack` *generated files only with native datatypes.*

`-L` or `--latest`
Use latest version of the HDF5 file format.

`-c` *max_compact_links* or `--compact`=*max_compact_links*
Set the maximum number of links, *max_compact_links*, that can be stored in a group header message (compact format).

`-d` *min_indexed_links* or `--indexed`=*min_indexed_links*
Set the minimum number of links, *min_indexed_links*, in the indexed format.

max_compact_links and *min_indexed_links* are closely related and the first must be equal to or greater than the second. In the general case, however, performance will suffer, possibly dramatically, if they are equal; performance can be improved by tuning the gap between the two values to minimize unnecessary thrashing between the compact storage and indexed storage modes as group size waxes and wanes. The relationship between *max_compact_links* and *min_indexed_links* is most important when group sizes are highly dynamic; that relationship is much less important in files with a stable structure. Compact mode is space and performance-efficient when groups have small numbers of members; indexed mode requires slightly more storage space, but provides increasingly better performance as the number of members in each group increases.

`-m` *number* or `--threshold`=*number*
Apply filter(s) only to objects whose size in bytes is equal to or greater than *number*. If no size is specified, a threshold of 1024 bytes is assumed.

`-s` *min_size*[:*header_type*] or `--ssize`=*min_size*[:*header_type*]
Set the minimum size of optionally specified types of shared object header messages.

min_size is the minimum size, in bytes, of a shared object header message. Header messages smaller than the specified size will not be shared.

header_type specifies the type(s) of header message that this minimum size is to be applied to. Valid values of *header_type* are any of the following:

`dspace`	for dataspace header messages
`dtype`	for datatype header messages
`fill`	for fill values
`pline`	for property list header messages
`attr`	for attribute header messages

If *header_type* is not specified, *min_size* will be applied to all header messages.

`-f` *filter* or `--filter`=*filter*

Filter type

filter is a string of the following format:

list_of_objects : *name_of_filter*[=*filter_parameters*]

list_of_objects is a comma separated list of object names meaning apply the filter(s) only to those objects. If no object names are specified, the filter is applied to all objects.

name_of_filter can be one of the following:
`GZIP`, to apply the HDF5 GZIP filter (GZIP compression)
`SZIP`, to apply the HDF5 SZIP filter (SZIP compression)
`SHUF`, to apply the HDF5 shuffle filter
`FLET`, to apply the HDF5 checksum filter
`NBIT`, to apply the HDF5 N-bit filter
`SOFF`, to apply the HDF5 scale/offset filter
`NONE`, to remove any filter(s)

filter_parameters conveys optional compression information:
`GZIP=`*deflation_level* from 1-9
`SZIP=`*pixels_per_block,coding_method*
pixels_per_block is a even number in the range 2-32.
coding_method is `EC` or `NN`.
`SHUF` (no parameter)
`FLET` (no parameter)
`NBIT` (no parameter)
`SOFF=`*scale_factor,scale_type*
scale_factor is an integer.
scale_type is either `IN` or `DS`.
`NONE` (no parameter)

`-l` *layout* or `--layout`=*layout*

Layout type

layout is a string of the following format:

list_of_objects : *layout_type*[=*layout_parameters*]

list_of_objects is a comma separated list of object names, meaning that layout information is supplied for those objects. If no object names are specified, the layout is applied to all objects.

layout_type can be one of the following:
`CHUNK`, to apply chunking layout
`COMPA`, to apply compact layout
`CONTI`, to apply continuous layout

layout_parameters is present only in the `CHUNK` case and specifies the chunk size of each dimension in the following format with no intervening spaces:

```
dim_1 × dim_2 × ... dim_n
```

`-e` *file*
File containing the -f and -l options (only filter and layout flags)

in_file
Input HDF5 file

out_file
Output HDF5 file

Examples:

1. `h5repack -f GZIP=1 -v file1 file2`
   Applies GZIP compression to all objects in `file1` and saves the output in `file2`. Prints verbose output.

2. `h5repack -f dset1:SZIP=8,NN file1 file2`
   Applies SZIP compression only to object `dset1`.

3. `h5repack -l dset1,dset2:CHUNK=20x10 file1 file2`
   Applies chunked layout to objects `dset1` and `dset2`.

History:

Release	**Command Line Tool**
1.6.2	`h5repack` introduced in this release.
1.8.0	`h5repack` command line syntax changed in this release.
1.8.1	Original syntax restored; both the new and the original syntax are now supported.

Tool Name: h5repart

Syntax:

`h5repart [-v] [-V] [-[b|m]`*N*`[g|m|k]]` [-family_to_sec2] *source_file dest_file*

Purpose:

Repartitions a file or family of files.

Description:

`h5repart` joins a family of files into a single file, or copies one family of files to another while changing the size of the family members. `h5repart` can also be used to copy a single file to a single file with holes. At this stage, `h5repart` can not split a single non-family file into a family of file(s).

To convert a family of file(s) to a single non-family file (`sec2` file), the option `-family_to_sec2` has to be used.

Sizes associated with the `-b` and `-m` options may be suffixed with `g` for gigabytes, `m` for megabytes, or `k` for kilobytes.

File family names include an integer `printf` format such as `%d`.

Options and Parameters:

`-v`	Produce verbose output.
`-V`	Print a version number and exit.
`-b`*N*	The I/O block size, defaults to 1kB
`-m`*N*	The destination member size or 1GB
`-family_to_sec2`	Convert file driver from family to sec2
source_file	The name of the source file
dest_file	The name of the destination files

Tool Name: h5jam/h5unjam

Syntax:

```
h5jam -u user_block -i in_file.h5 [-o out_file.h5] [--clobber]
h5jam -h

h5unjam -i in_file.h5 [-u user_block | --delete] [-o out_file.h5]
h5unjam -h
```

Purpose:

Adds user block to front of an HDF5 file, to create a new concatenated file.
Splits user block and HDF5 file into two files: user block data and HDF5 data.

Description:

`h5jam` concatenates a `user_block` file and an HDF5 file to create an HDF5 file with a user block. The user block can be either binary or text. The output file is padded so that the HDF5 header begins on byte 512, 1024, etc.. (See the HDF5 File Format.)

If `out_file.h5` is given, a new file is created with the `user_block` followed by the contents of `in_file.h5`. In this case, `infile.h5` is unchanged.

If `out_file.h5` is not specified, the `user_block` is added to `in_file.h5`.

If `in_file.h5` already has a user block, the contents of `user_block` will be added to the end of the existing user block, and the file shifted to the next boundary. If `--clobber` is set, any existing user block will be overwritten.

`h5unjam` splits an HDF5 file, writing the user block to a file or to stdout and the HDF5 file to an HDF5 file with a header at byte zero (`0`, i.e., with no user block).

If `out_file.h5` is given, a new file is created with the contents of `in_file.h5` without the user block. In this case, `infile.h5` is unchanged.

If `out_file.h5` is not specified, the `user_block` is removed and `in_file.h5` is rewritten, starting at byte 0.

If `user_block` is set, the user block will be written to `user_block`. If `user_block` is not set, the user block, if any, will be written to stdout. If `--delete` is selected, the user block will not be written.

Examples:

Create new file, `newfile.h5`, with the text in file `mytext.txt` as the user block for the HDF5 file `file.h5`.

```
h5jam -u mytext.txt -i file.h5 -o newfile.h5
```

Add text in file `mytext.txt` to front of HDF5 dataset, `file.h5`.

```
h5jam -u mytext.txt -i file.h5
```

Overwrite the user block, if any, in `file.h5` with the contents of `mytext.txt`.

```
h5jam -u mytext.txt -i file.h5 --clobber
```

For an HDF5 file, `with_ub.h5`, with a user block, extract the user block to `user_block.txt` and the HDF5 portion of the file to `wo_ub.h5`.

```
    h5unjam -i with_ub.h5 -u user_block.txt -o wo_ub.h5
```

Return Value:

`h5jam` returns the size of the output file, or -1 if an error occurs.

`h5unjam` returns the size of the output file, or -1 if an error occurs.

Caveats

These tools copy all the data sequentially in the file(s) to new offsets. For a large file, this copy will take a long time.

The most efficient way to create a user block is to create the file with a user block (see `H5Pset_user_block`), and write the user block data into that space from a program.

The user block is completely opaque to the HDF5 library and to the `h5jam` and `h5unjam` tools. The user block is simply read or written as a string of bytes, which could be text or any kind of binary data; it is up to the user to know what the contents of the user block means and how to process it.

When the user block is extracted, all the data is written to the output, including any padding or unwritten data.

This tool moves the HDF5 portion of the file through byte copies; i.e., it does not read or interpret the HDF5 objects.

Tool Name: h5copy

Syntax:

`h5copy [OPTIONS] [OBJECTS]`

Purpose:

Copy an object from one HDF5 file to another HDF5 file.

Description:

`h5copy` copies an HDF5 object (dataset, named datatype, or group) from one HDF5 file to another HDF5 file, which may either already exist or not.

Arguments:

Options and Parameters:

`-h` or `--help`

Print a usage message and exit.

`-v` or `--verbose`

Produce verbose output, printing information regarding the specified options and objects.

`-f flag_type` or `--flag=flag_type`

Specify one or more of several copy options; `flag_type` may be one of the following strings or a logical AND of two or more:

`shallow`	Copy only immediate members of a group. *(Default:* Recursively copy all objects below the group.*)*
`soft`	Expand soft links to copy target objects. *(Default:* Keep soft links as they are.*)*
`ext`	Expand external links to copy external objects. *(Default:* Keep external links as they are.*)*
`ref`	Copy objects that are pointed to by references. *(Default:* Update only the values of object references.*)*
`attr`	Copy objects without copying attributes. *(Default:* Copy objects and all attributes.*)*
`allflags`	Switch each setting above from the default to the setting described in this table. Equivalent to logical AND of all flags above.

`-V` or `--Version`

Print version information.

Objects (all required):

`-i input_file` or `--input=input_file`

Input HDF5 file name

`-o output_file` or `--output=output_file`

Output HDF5 file name (existing or non-existing)

`-s source_object` or `--source=source_object`

Input HDF5 object name within the source file

`-d destination_object` or `--destination=destination_object`

Output HDF5 object name within the destination file

Example Usage

In verbose mode, create a new file, `test1.out.h5`, containing the object `array` in the root group, copied from the existing file `test1.h5` and object `array`.

```
h5copy -v -i "test1.h5" -o "test1.out.h5" -s "/array" -d "/array
```

In verbose mode and using the flag `shallow` to prevent recursion in the file hierarchy, create a new file, `test1.out.h5`, containing the object `array` in the root group, copied from the existing file `test1.h5` and object `array`.

```
    h5copy -v -f shallow -i "test1.h5" -s "/array" -o test1.out.h5" -d "/array"
```

History:

Release	**Command Line Tool**
1.8.0	Tool introduced in this release.

Tool Name: h5mkgrp

Syntax:

`h5mkgrp [OPTIONS]` *`file_name group_name`*`...`

Purpose:

Creates new group(s) in an HDF5 file.

Description:

`h5mkgrp` creates one or more new groups in an HDF5 file.

Options and Parameters:

`file_name`

Name of HDF5 file within which new group is to be created.

`group_name`

Name of group to be created; specified as full path name from the root group, i.e., starting with a slash (/).

Options:

`-h, --help`

Print a usage message and exit.

`-l, --latest`

Use latest version of file format to create new group.

`-p, --parents`

Create parent or intervening groups as needed. Issue no error if intervening groups or new group already exist.

`-v, --verbose`

Print verbose output, including information about file, group(s), and options.

`-V, --version`

Print tool version number then exit. Tool version number is that of the corresponding HDF5 Library.

Example Usage

Create a new group, `new_group`, within the existing group `/a/b` in the file `HDF5_file`.

```
h5mkgrp "HDF5_file" "/a/b/new_group"
```

Create a new group, `new_group`, within the group `/a/b` in the file `HDF5_file`. Create the groups `a` and `b` if they do not already exist. Issue no error if the intervening groups or the new group already exist.

```
h5mkgrp -p "HDF5_file" "/a/b/new_group"
```

Create the new groups `/a/b/new_c` and `/a/x/new_4` in the file `HDF5_file`. The groups `/a/b` and `/a/x` must already exist.

```
h5mkgrp -p "HDF5_file" "/a/b/new_c" "/a/x/new_4"
```

History:

Release	**Command Line Tool**
1.8.0	Tool introduced in this release.

Tool Name: h5import

Syntax:

```
h5import infile in_options [infile in_options ...] -o outfile
h5import infile in_options [infile in_options ...] -outfile outfile
h5import -h
h5import -help
```

Purpose:

Imports data into an existing or new HDF5 file.

Description:

`h5import` converts data from one or more ASCII or binary files, *`infile`*, into the same number of HDF5 datasets in the existing or new HDF5 file, *`outfile`*. Data conversion is performed in accordance with the user-specified type and storage properties specified in *`in_options`*.

The primary objective of `h5import` is to import floating point or integer data. The utility's design allows for future versions that accept ASCII text files and store the contents as a compact array of one-dimensional strings, but that capability is not implemented in HDF5 Release 1.6.

Input data and options:

Input data can be provided in one of the following forms:

- ◊ As an ASCII, or plain-text, file containing either floating point or integer data
- ◊ As a binary file containing either 32-bit or 64-bit native floating point data
- ◊ As a binary file containing native integer data, signed or unsigned and 8-bit, 16-bit, 32-bit, or 64-bit.
- ◊ As an ASCII, or plain-text, file containing text data. (This feature is not implemented in HDF5 Release 1.6.)

Each input file, *`infile`*, contains a single *n*-dimensional array of values of one of the above types expressed in the order of fastest-changing dimensions first.

Floating point data in an ASCII input file may be expressed either in the fixed-point form (e.g., 323.56) or in scientific notation (e.g., 3.23E+02) in an ASCII input file.

Each input file can be associated with options specifying the datatype and storage properties. These options can be specified either as *command line arguments* or in a *configuration file*. Note that exactly one of these approaches must be used with a single input file.

Command line arguments, best used with simple input files, can be used to specify the class, size, dimensions of the input data and a path identifying the output dataset.

The recommended means of specifying input data options is in a configuration file; this is also the only means of specifying advanced storage features. See further discussion in "The configuration file" below.

The only required option for input data is dimension sizes; defaults are available for all others.

`h5import` will accept up to 30 input files in a single call. Other considerations, such as the maximum length of a command line, may impose a more stringent limitation.

Output data and options:

The name of the output file is specified following the `-o` or `-output` option in *`outfile`*. The data from each input file is stored as a separate dataset in this output file. *`outfile`* may be an existing file. If it does not yet exist, `h5import` will create it.

Output dataset information and storage properties can be specified only by means of a configuration file.

Dataset path	If the groups in the path leading to the dataset do not exist, `h5import` will create them. If no group is specified, the dataset will be created as a member of the root group. If no dataset name is specified, the default name is `dataset0` for the first input dataset, `dataset1` for the second input dataset, `dataset2` for the third input dataset, etc. `h5import` does not overwrite a pre-existing dataset of the specified or default name. When an existing dataset of a conflicting name is encountered, `h5import` quits with an error; the current input file and any subsequent input files are not processed.
Output type	Datatype parameters for output data
Output data class	Signed or unsigned integer or floating point
Output data size	8-, 16-, 32-, or 64-bit integer 32- or 64-bit floating point

Output architecture	`IEEE`
	`STD`
	`NATIVE` (Default)
	Other architectures are included in the `h5import` design but are not implemented in this release.

Output byte order	Little- or big-endian. Relevant only if output architecture is `IEEE`, `UNIX`, or `STD`; fixed for other architectures.
Dataset layout and storage properties	Denote how raw data is to be organized on the disk. If none of the following are specified, the default configuration is contiguous layout and with no compression.
Layout	Contiguous (Default) Chunked
External storage	Allows raw data to be stored in a non-HDF5 file or in an external HDF5 file. Requires contiguous layout.
Compressed	Sets the type of compression and the level to which the dataset must be compressed. Requires chunked layout.
Extendable	Allows the dimensions of the dataset increase over time and/or to be unlimited. Requires chunked layout.
Compressed and extendable	Requires chunked layout.

Command-line arguments:
The `h5import` syntax for the command-line arguments, *`in_options`*, is as follows:

```
h5import infile -d dim_list [-p pathname] [-t input_class] [-s
input_size] [infile ...] -o outfile
```
or
```
h5import infile -dims dim_list [-path pathname] [-type input_class]
[-size input_size] [infile ...] -outfile outfile
```
or
```
h5import infile -c config_file [infile ...] -outfile outfile
```

Note the following: If the `-c` *`config_file`* option is used with an input file, no other argument can be used with that input file. If the `-c` *`config_file`* option is not used with an input data file, the `-d` *`dim_list`* argument (or `-dims` *`dim_list`*) must be used and any combination of the remaining options may be used. Any arguments used must appear in *exactly* the order used in the syntax declarations immediately above.

The configuration file:
A configuration file is specified with the `-c` *`config_file`* option:

```
h5import infile -c config_file [infile -c config_file2 ...] -outfile
outfile
```

The configuration file is an ASCII file and must be organized as "Configuration_Keyword Value" pairs, with one pair on each line. For example, the line indicating that the input data class (configuration keyword `INPUT-CLASS`) is floating point in a text file (value `TEXTFP`) would appear as follows:

```
INPUT-CLASS TEXTFP
```

A configuration file may have the following keywords each followed by one of the following defined values. One entry for each of the first two keywords, `RANK` and `DIMENSION-SIZES`, is required; all other keywords are optional.

Keyword **Value**	**Description**
`RANK`	The number of dimensions in the dataset. (Required)
`rank`	An integer specifying the number of dimensions in the dataset. Example: `4` for a 4-dimensional dataset.
`DIMENSION-SIZES`	Sizes of the dataset dimensions. (Required)
`dim_sizes`	A string of space-separated integers specifying the sizes of the dimensions in the dataset. The number of sizes in this entry must match the value in the `RANK` entry. The fastest-changing dimension must be listed first. Example: `4 3 4 38` for a 38x4x3x4 dataset.

PATH	Path of the output dataset.
path	The full HDF5 pathname identifying the output dataset relative to the root group within the output file. I.e., *path* is a string consisting of optional group names, each followed by a slash, and ending with a dataset name. If the groups in the path do no exist, they will be created. If PATH is not specified, the output dataset is stored as a member of the root group and the default dataset name is dataset0 for the first input dataset, dataset1 for the second input dataset, dataset2 for the third input dataset, etc. Note that h5import does not overwrite a pre-existing dataset of the specified or default name. When an existing dataset of a conflicting name is encountered, h5import quits with an error; the current input file and any subsequent input files are not processed. Example: The configuration file entry `PATH grp1/grp2/dataset1` indicates that the output dataset dataset1 will be written in the group grp2/ which is in the group grp1/, a member of the root group in the output file.
INPUT-CLASS	A string denoting the type of input data.
TEXTIN	Input is signed integer data in an ASCII file.
TEXTUIN	Input is unsigned integer data in an ASCII file.
TEXTFP	Input is floating point data in either fixed-point notation (e.g., 325.34) or scientific notation (e.g., 3.2534E+02) in an ASCII file.
IN	Input is signed integer data in a binary file.
UIN	Input is unsigned integer data in a binary file.
FP	Input is floating point data in a binary file. (Default)
STR	Input is character data in an ASCII file. With this value, the configuration keywords RANK, DIMENSION-SIZES, OUTPUT-CLASS, OUTPUT-SIZE, OUTPUT-ARCHITECTURE, and OUTPUT-BYTE-ORDER will be ignored. (Not implemented in this release.)
INPUT-SIZE	An integer denoting the size of the input data, in bits.
8 16 32 64	For signed and unsigned integer data: TEXTIN, TEXTUIN, IN, or UIN. (Default: 32)
32 64	For floating point data: TEXTFP or FP. (Default: 32)

`OUTPUT-CLASS`	A string denoting the type of output data.
`IN`	Output is signed integer data. (Default if `INPUT-CLASS` is `IN` or `TEXTIN`)
`UIN`	Output is unsigned integer data. (Default if `INPUT-CLASS` is `UIN` or `TEXTUIN`)
`FP`	Output is floating point data. (Default if `INPUT-CLASS` is not specified or is `FP` or `TEXTFP`)
`STR`	Output is character data, to be written as a 1-dimensional array of strings. (Default if `INPUT-CLASS` is `STR`) (Not implemented in this release.)
`OUTPUT-SIZE`	An integer denoting the size of the output data, in bits.

`8`
`16`
`32`
`64`
	For signed and unsigned integer data: `IN` or `UIN`. (Default: Same as `INPUT-SIZE`, else `32`)

`32`
`64`
	For floating point data: `FP`. (Default: Same as `INPUT-SIZE`, else `32`)

`OUTPUT-ARCHITECTURE`	A string denoting the type of output architecture.

`NATIVE`
`STD`
`IEEE`
`INTEL` *
`CRAY` *
`MIPS` *
`ALPHA` *
`UNIX` *
	See the "Predefined Atomic Types" section in the "HDF5 Datatypes" chapter of the *HDF5 User's Guide* for a discussion of these architectures.
	Values marked with an asterisk (*) are not implemented in this release.
	(Default: `NATIVE`)

`OUTPUT-BYTE-ORDER`	A string denoting the output byte order. This entry is ignored if the `OUTPUT-ARCHITECTURE` is not specified or if it is not specified as `IEEE`, `UNIX`, or `STD`.
`BE`	Big-endian. (Default)
`LE`	Little-endian.

The following options are disabled by default, making the default storage properties no chunking, no compression, no external storage, and no extensible dimensions.

`CHUNKED-DIMENSION-SIZES`	Dimension sizes of the chunk for chunked output data.
`chunk_dims`	A string of space-separated integers specifying the dimension sizes of the chunk for chunked output data. The number of dimensions must correspond to the value of `RANK`. The presence of this field indicates that the output dataset is to be stored in chunked layout; if this configuration field is absent, the dataset will be stored in contiguous layout.
`COMPRESSION-TYPE`	Type of compression to be used with chunked storage. Requires that `CHUNKED-DIMENSION-SIZES` be specified.
`GZIP`	Gzip compression. Other compression algorithms are not implemented in this release of `h5import`.
`COMPRESSION-PARAM`	Compression level. Required if `COMPRESSION-TYPE` is specified.
`1` through `9`	Gzip compression levels: `1` will result in the fastest compression while `9` will result in the best compression ratio. (Default: 6. The default gzip compression level is 6; not all compression methods will have a default level.)
`EXTERNAL-STORAGE`	Name of an external file in which to create the output dataset. Cannot be used with `CHUNKED-DIMENSIONS-SIZES`, `COMPRESSION-TYPE`, OR `MAXIMUM-DIMENSIONS`.
`external_file`	A string specifying the name of an external file.
`MAXIMUM-DIMENSIONS`	Maximum sizes of all dimensions. Requires that `CHUNKED-DIMENSION-SIZES` be specified.
`max_dims`	A string of space-separated integers specifying the maximum size of each dimension of the output dataset. A value of `-1` for any dimension implies unlimited size for that particular dimension. The number of dimensions must correspond to the value of `RANK`.

Options and Parameters:

`infile(s)`	Name of the Input file(s).
`in_options`	Input options. Note that while only the `-dims` argument is required, arguments must used in the order in which they are listed below.
`-d` *`dim_list`*	
`-dims` *`dim_list`*	Input data dimensions. *`dim_list`* is a string of comma-separated numbers with no spaces describing the dimensions of the input data. For example, a 50 x 100 2-dimensional array would be specified as `-dims 50,100`. Required argument: if no configuration file is used, this command-line argument is mandatory.
`-p` *`pathname`*	
`-pathname` *`pathname`*	*`pathname`* is a string consisting of one or more strings separated by slashes (`/`) specifying the path of the dataset in the output file. If the groups in the path do no exist, they will be created. Optional argument: if not specified, the default path is `dataset1` for the first input dataset, `dataset2` for the second input dataset, `dataset3` for the third input dataset, etc. `h5import` does not overwrite a pre-existing dataset of the specified or default name. When an existing dataset of a conflicting name is encountered, `h5import` quits with an error; the current input file and any subsequent input files are not processed.
`-t` *`input_class`*	
`-type` *`input_class`*	*`input_class`* specifies the class of the input data and determines the class of the output data. Valid values are as defined in the Keyword/Values table in the section "The configuration file" above. Optional argument: if not specified, the default value is `FP`.
`-s` *`input_size`*	
`-size` *`input_size`*	*`input_size`* specifies the size in bits of the input data and determines the size of the output data. Valid values for signed or unsigned integers are `8`, `16`, `32`, and `64`. Valid values for floating point data are `32` and `64`. Optional argument: if not specified, the default value is `32`.
`-c` *`config_file`*	*`config_file`* specifies a configuration file. This argument replaces all other arguments except *`infile`* and `-o` *`outfile`*
`-h`	

`-help`	Prints the `h5import` usage summary:
	`h5import -h[elp], OR`
	`h5import <infile> <options> [<infile> <options>...] -o[utfile] <outfile>`
	Then exits.

`outfile`	Name of the HDF5 output file.

Examples:

Using command-line arguments:

```
h5import infile -dims 2,3,4 -type TEXTIN -size 32 -o out1
```

This command creates a file `out1` containing a single 2x3x4 32-bit integer dataset. Since no pathname is specified, the dataset is stored in `out1` as `/dataset1`.

```
h5import infile -dims 20,50 -path bin1/dset1 -type FP -size 64 -o out2
```

This command creates a file `out2` containing a single a 20x50 64-bit floating point dataset. The dataset is stored in `out2` as `/bin1/dset1`.

Sample configuration files:

The following configuration file specifies the following:
– The input data is a 5x2x4 floating point array in an ASCII file.
– The output dataset will be saved in chunked layout, with chunk dimension sizes of 2x2x2.
– The output datatype will be 64-bit floating point, little-endian, IEEE.
– The output dataset will be stored in *`outfile`* at `/work/h5/pkamat/First-set`.
– The maximum dimension sizes of the output dataset will be 8x8x(unlimited).

```
PATH work/h5/pkamat/First-set
INPUT-CLASS TEXTFP
RANK 3
DIMENSION-SIZES 5 2 4
OUTPUT-CLASS FP
OUTPUT-SIZE 64
OUTPUT-ARCHITECTURE IEEE
OUTPUT-BYTE-ORDER LE
CHUNKED-DIMENSION-SIZES 2 2 2
MAXIMUM-DIMENSIONS 8 8 -1
```

The next configuration file specifies the following:
– The input data is a 6x3x5x2x4 integer array in a binary file.
– The output dataset will be saved in chunked layout, with chunk dimension sizes of 2x2x2x2x2.
– The output datatype will be 32-bit integer in `NATIVE` format (as the output architecture is not specified).
– The output dataset will be compressed using Gzip compression with a compression level of 7.
– The output dataset will be stored in *`outfile`* at `/Second-set`.

```
PATH Second-set
INPUT-CLASS IN
RANK 5
DIMENSION-SIZES 6 3 5 2 4
OUTPUT-CLASS IN
OUTPUT-SIZE 32
CHUNKED-DIMENSION-SIZES 2 2 2 2 2
COMPRESSION-TYPE GZIP
COMPRESSION-PARAM 7
```

History:

Release	**Command Line Tool**
1.6.0	Tool introduced in this release.

Tool Name: gif2h5

Syntax:

`gif2h5` *gif_file h5_file*

Purpose:

Converts a GIF file to an HDF5 file.

Description:

`gif2h5` accepts as input the GIF file *gif_file* and produces the HDF5 file *h5_file* as output.

Options and Parameters:

gif_file	The name of the input GIF file
h5_file	The name of the output HDF5 file

Tool Name: h52gif
Syntax:

`h52gif` *h5_file gif_file* `-i` *h5_image* `[-p` *h5_palette*`]`

Purpose:

Converts an HDF5 file to a GIF file.

Description:

`h52gif` accepts as input the HDF5 file *h5_file* and the names of images and associated palettes within that file as input and produces the GIF file *gif_file*, containing those images, as output.

`h52gif` expects *at least* one *h5_image*. You may repeat
`-i` *h5_image* `[-p` *h5_palette*`]`
up to 50 times, for a maximum of 50 images.

Options and Parameters:

h5_file	The name of the input HDF5 file
gif_file	The name of the output GIF file
`-i` *h5_image*	Image option, specifying the name of an HDF5 image or dataset containing an image to be converted
`-p` *h5_palette*	Palette option, specifying the name of an HDF5 dataset containing a palette to be used in an image conversion

Tool Name: h5toh4

Syntax:

`h5toh4 -h`
`h5toh4` *h5file h4file*
`h5toh4` *h5file*
`h5toh4 -m` *h5file1 h5file2 h5file3 ...*

Purpose:

Converts an HDF5 file into an HDF4 file.

Description:

`h5toh4` is an HDF5 utility which reads an HDF5 file, *h5file*, and converts all supported objects and pathways to produce an HDF4 file, *h4file*. If *h4file* already exists, it will be replaced.

If only one file name is given, the name must end in `.h5` and is assumed to represent the HDF5 input file. `h5toh4` replaces the `.h5` suffix with `.hdf` to form the name of the resulting HDF4 file and proceeds as above. If a file with the name of the intended HDF4 file already exists, `h5toh4` exits with an error without changing the contents of any file.

The `-m` option allows multiple HDF5 file arguments. Each file name is treated the same as the single file name case above.

The `-h` option causes the following syntax summary to be displayed:

```
h5toh4 file.h5 file.hdf
h5toh4 file.h5
h5toh4 -m file1.h5 file2.h5 ...
```

The following HDF5 objects occurring in an HDF5 file are converted to HDF4 objects in the HDF4 file:

- ◊ HDF5 group objects are converted into HDF4 Vgroup objects. HDF5 hard links and soft links pointing to objects are converted to HDF4 Vgroup references.
- ◊ HDF5 dataset objects of integer datatype are converted into HDF4 SDS objects. These datasets may have up to 32 fixed dimensions. The slowest varying dimension may be extendable. 8-bit, 16-bit, and 32-bit integer datatypes are supported.
- ◊ HDF5 dataset objects of floating point datatype are converted into HDF4 SDS objects. These datasets may have up to 32 fixed dimensions. The slowest varying dimension may be extendable. 32-bit and 64-bit floating point datatypes are supported.
- ◊ HDF5 dataset objects of single dimension and compound datatype are converted into HDF4 Vdata objects. The length of that single dimension may be fixed or extendable. The members of the compound datatype are constrained to be no more than rank 4.
- ◊ HDF5 dataset objects of single dimension and fixed length string datatype are converted into HDF4 Vdata objects. The HDF4 Vdata is a single field whose order is the length of the HDF5 string type. The number of records of the Vdata is the length of the single dimension which may be fixed or extendable.

Other objects are not converted and are not recorded in the resulting *h4file*.

Attributes associated with any of the supported HDF5 objects are carried over to the HDF4 objects. Attributes may be of integer, floating point, or fixed length string datatype and they may have up to 32 fixed dimensions.

All datatypes are converted to big-endian. Floating point datatypes are converted to IEEE format.

Note:

The `h5toh4` and `h4toh5` utilities are no longer part of the HDF5 product; they are distributed separately through the page Converting between HDF (4.x) and HDF5.

Options and Parameters:

`-h`	Displays a syntax summary.
`-m`	Converts multiple HDF5 files to multiple HDF4 files.
h5file	The HDF5 file to be converted.
h4file	The HDF4 file to be created.

Tool Name: h4toh5

Syntax:

`h4toh5 -h`
`h4toh5` *h4file h5file*
`h4toh5` *h4file*

Purpose:

Converts an HDF4 file to an HDF5 file.

Description:

`h4toh5` is a file conversion utility that reads an HDF4 file, *h4file* (`input.hdf` for example), and writes an HDF5 file, *h5file* (`output.h5` for example), containing the same data.

If no output file *h5file* is specified, `h4toh5` uses the input filename to designate the output file, replacing the extension `.hdf` with `.h5`. For example, if the input file `scheme3.hdf` is specified with no output filename, `h4toh5` will name the output file `scheme3.h5`.

The `-h` option causes a syntax summary similar to the following to be displayed:

```
h4toh5 inputfile.hdf outputfile.h5
h4toh5 inputfile.hdf
```

Each object in the HDF4 file is converted to an equivalent HDF5 object, according to the mapping described in *Mapping HDF4 Objects to HDF5 Objects*.

`h4toh5` converts the following HDF4 objects:

HDF4 Object	Resulting HDF5 Object
SDS	Dataset
GR, RI8, and RI24 image	Dataset
Vdata	Dataset
Vgroup	Group
Annotation	Attribute
Palette	Dataset

Note:

The `h4toh5` and `h5toh4` utilities are no longer part of the HDF5 product; they are distributed separately through the page Converting between HDF (4.x) and HDF5.

Options and Parameters:

`-h`	Displays a syntax summary.
h4file	The HDF4 file to be converted.
h5file	The HDF5 file to be created.

Tool Name: h5stat

Syntax:

`h5stat` [*OPTIONS*] *file*

Purpose:

Reports HDF5 file and object statistics.

Description:

`h5stat` reports statistics regarding an HDF5 file and the objects in that file.

See “RFC: h5stat tool” for a complete description of this tool.

Options and Parameters:

`-V` or `--version`	Print version number and exit.
`-f` or `--file`	Print file information.
`-F` or `--filemetadata`	Print file meta data.
`-g` or `--group`	Print group information.
`-G` or `--groupmetadata`	Print group meta data.
`-d` or `--dset`	Print dataset information.
`-D` or `--dsetmetadata`	Print dataset meta data.
`-T` or `--dtypemetadata`	Print datatype meta data.
`-A` or `--attribute`	Print attribute information.

History:

Release	**Command Line Tool**
1.8.0	Tool introduced in this release.

Tool Name: h5perf
Syntax:

`h5perf` [`-h` | `--help`]
`h5perf` [*options*]

Purpose:

Tests Parallel HDF5 performance.

Description:

`h5perf` is a tool for testing the performance of the Parallel HDF5 Library. The tool can perform testing with 1-dimensional and 2-dimensional buffers and datasets. For details regarding data organization and access, see "h5perf, a Parallel File System Benchmarking Tool."

The following environment variables have the following effects on `h5perf` behavior:

`HDF5_NOCLEANUP`	If set, `h5perf` does not remove data files. (*Default:* Data files are removed.)
`HDF5_MPI_INFO`	Must be set to a string containing a list of semi-colon separated `key=value` pairs for the MPI `INFO` object. Example:
`HDF5_PARAPREFIX`	Sets the prefix for parallel output data files.

Options and Parameters:

These terms are used as follows in this section:

file	A filename
size	A size specifier, expressed as an integer greater than or equal to 0 (zero) followed by a size indicator: `K` for kilobytes (1024 bytes) `M` for megabytes (1048576 bytes) `G` for gigabytes (1073741824 bytes) Example: `37M` specifies 37 megabytes or 38797312 bytes.
N	An integer greater than or equal to 0 (zero)

`-h`, `--help`

Prints a usage message and exits.

`-a` *size*, `--align=`*size*

Specifies the alignment of objects in the HDF5 file.
(*Default:* `1`)

`-A` *api_list*, `--api=`*api_list*

Specifies which APIs to test. *api_list* is a comma-separated list with the following valid values:

`phdf5`	Parallel HDF5
`mpiio`	MPI-I/O
`posix`	POSIX

(*Default:* All APIs)

Example, `--api=mpiio,phdf5` specifies that the MPI I/O and Parallel HDF5 APIs are to be monitored.

`-B` *size*, `--block-size=`*size*

Controls the block size within the transfer buffer.
(*Default:* Half the number of bytes per process per dataset)

Block size versus transfer buffer size:
The *transfer buffer size* is the size of a buffer in memory. The data in that buffer is broken into *block size* pieces and written to the file.

Transfer buffer size is discussed below with the `-x (or --min-xfer-size) and -X (or --max-xfer-size) options.`

The pattern in which the blocks are written to the file is described in the discussion of the `-I` (or `--interleaved`) option.

`-c`, `--chunk`

Creates HDF5 datasets in chunked layout.
(*Default:* Off)

`-C`, `--collective`

Use collective I/O for the MPI I/O and Parallel HDF5 APIs.
(*Default:* Off, i.e., independent I/O)

If this option is set and the MPI-I/O and PHDF5 APIs are in use, all the blocks of every process will be written at once with an MPI derived type.

`-d` *N*, `--num-dsets`*N*

Sets the number of datasets per file.
(*Default:* `1`)

`-D` *debug_flags*, `--debug=`*debug_flags*

Sets the debugging level. *debug_flags* is a comma-separated list of debugging flags with the following valid values:

`1`	Minimal debugging
`2`	Moderate debugging (“not quite everything”)
`3`	Extensive debugging (“everything”)
`4`	All possible debugging (“the kitchen sink”)
`r`	Raw data I/O throughput information
`t`	Times, in additions to throughputs
`v`	Verify data correctness

(*Default:* No debugging)

Example: `--debug=2,r,t` specifies to run a moderate level of debugging while collecting raw data I/O throughput information and verifying the correctness of the data.

Throughput values are computed by dividing the total amount of transferred data (excluding metadata) over the time spent by the slowest process. Several time counters are defined to measure the data transfer time and the total elapsed time; the latter includes the time spent during file open and close operations. A number of iterations can be specified with the option `-i` (or `--num-iterations`) to create the desired population of measurements from which maximum, minimum, and average values can be obtained. The timing scheme is the following:

```
    for each iteration
```

```
        initialize elapsed time counter
        initialize data transfer time counter
        for each file
            start and accumulate elapsed time counter
                file open
                start and accumulate data transfer time counter
                    access entire file
                stop data transfer time counter
                file close
            stop elapsed time counter
        end file
        save elapsed time counter
        save data transfer time counter
    end iteration
```

The reported write throughput is based on the accumulated data transfer time, while the write open-close throughput uses the accumulated elapsed time.

`-e` *size*, `--num-bytes=`*size*

Specifies the number of bytes per process per dataset.
(*Default:* `256K` for 1D, `8K` for 2D)

Depending on the selected geometry, each test dataset can be a linear array of size *bytes-per-process * num-processes* or a square array of size *(bytes-per-process * num-processes)* × *(bytes-per-process * num-processes)*. The number of processes is set by the `-p` (or `--min-num-processes`) and `-P` (or `--max-num-processes`) options.

`-F` *N*, `--num-files=`*N*

Specifies the number of files.
(*Default:* `1`)

`-g`, `--geometry`

Selects 2D geometry for testing.
(*Default:* Off, i.e., 1D geometry)

`-i` *N*, `--num-iterations=`*N*

Sets the number of iterations to perform.
(*Default:* `1`)

`-I`, `--interleaved`

Sets interleaved block I/O.
(*Default:* Contiguous block I/O)

Interleaved and contiguous patterns in 1D geometry:
When a contiguous access pattern is chosen, the dataset is evenly divided into *num-processes* regions and each process writes data to its assigned region. When interleaved blocks are written to a dataset, space for the first block of the first process is allocated in the dataset, then space is allocated for the first block of the second process, etc., until space is allocated for the first block of each process, then space is allocated for the second block of the first process, the second block of the second process, etc.

For example, with a three process run, 512KB bytes-per-process, 256KB transfer buffer size, and 64KB block size, each process must issue two transfer requests to complete access to the dataset.

Contiguous blocks of the first transfer request are written as follows:

```
1111----2222----3333----
```

Interleaved blocks of the first transfer request are written as follows:

```
123123123123------------
```

The actual number of I/O operations involved in a transfer request depends on the access pattern and communication mode. When using independent I/O with an interleaved access pattern, each process performs four small non-contiguous I/O operations per transfer request. If collective I/O is turned on, the combined content of the buffers of the three processes will be written using one collective I/O operation per transfer request.

For details regarding the impact of performance and access patterns in 2D, see “h5perf, a Parallel File System Benchmarking Tool.”

`-m`, `--mpi-posix`	Sets use of MPI-posix driver for HDF5 I/O. (*Default:* MPI-I/O driver)
`-n`, `--no-fill`	Specifies to not write fill values to HDF5 datasets. This option is supported only in HDF5 Release v1.6 or later. (*Default:* Off, i.e., write fill values)
`-o` *file*, `--output=`*file*	Sets the output file for raw data to *file*. (*Default:* None)
`-p` *N*, `--min-num-processes=`*N*	Sets the minimum number of processes to be used. (*Default:* `1`)
`-P` *N*, `--max-num-processes=`*N*	Sets the maximum number of processes to be used. (*Default:* All `MPI_COMM_WORLD` processes)
`-T` *size*, `--threshold=`*size*	Sets the threshold for alignment of objects in the HDF5 file. (*Default:* `1`)
`-w`, `--write-only`	Performs only write tests, not read tests. (*Default:* Read and write tests)

`-x` *size*, `--min-xfer-size=`*size*	Sets the minimum transfer buffer size. (*Default:* Half the number of bytes per processor per dataset) This option and the `-X` *size* option (or `--max-xfer-size=`*size*) control *transfer-buffer-size*, the size of the transfer buffer in memory. In 1D geometry, the transfer buffer is a linear array of size *transfer-buffer-size*. In 2D geometry, the transfer buffer is a rectangular array of size *block-size* × *transfer-buffer-size*, or *transfer-buffer-size* × *block-size* if the interleaved access pattern is selected.
`-X` *size*, `--max-xfer-size=`*size*	Sets the maximum transfer buffer size. (*Default:* The number of bytes per processor per dataset)

History:

Release	**Change**
1.6.0	Tool introduced in this release.
1.6.8 and 1.8.0	Option `-g`, `--geometry` introduced in this release.

Tool Name: h5perf_serial

Syntax:

`h5perf_serial` [`-h` | `--help`]
`h5perf_serial` [*options*]

Purpose:

Tests HDF5 serial performance.

Description:

`h5perf_serial` provides tools for testing the performance of the HDF5 Library in serial mode.

See "h5perf_serial, a Serial File System Benchmarking Tool" for a complete description of this tool.

The following environment variable can be set to control the specfied aspect of `h5perf_serial` behavior:

`HDF5_NOCLEANUP`	If set, `h5perf_serial` does not remove data files. *(Default: Data files are removed.)*
`HDF5_PREFIX`	Sets the prefix for output data files.

Options and Parameters:

The term *size specifier* is used as follows in this section:
A size specifier is an integer greater than or equal to 0 (zero) followed by a size indicator:
`K` for kilobytes (1024 bytes)
`M` for megabytes (1048576 bytes)
`G` for gigabytes (1073741824 bytes)
Example: `37M` specifies 37 megabytes or 38797312 bytes.

`-A` *api_list*	Specifies which APIs to test. *api_list* is a comma-separated list with the following valid values: `hdf5` HDF5 Library APIs `posix` POSIX APIs *(Default: All APIs are monitored.)* Example: `-A hdf5,posix` specifies that the HDF5 and POSIX APIs are to be monitored.
`-c` *chunk_size_list*	Specifies chunked storage and defines chunks dimensions and sizes. *(Default: Chunking is off.)* *chunk_size_list* is a comma-separated list of size specifiers. For example, a *chunk_size_list* value of `2K,4K,6M` specifies that chunking is turned on and that chunk size is 2 kilobytes by 4 kilobytes by 6 megabytes.
`-e` *dataset_size_list*	Specifies dataset dimensionality and dataset dimension sizes. *(Default dataset size is 100x200, or `100,200`.)* *dataset_size_list* is a comma-separated list of size specifiers, which are defined above.

	For example, a *dataset_size_list* value of `2K,4K,6M` specifies a 2 kilobytes by 4 kilobytes by 6 megabytes dataset.
`-i` *iterations*	Specifies the number of iterations to perform. *(Default: A single iteration, `1`, is performed.)* *iterations* is an integer specifying the number of iterations.
`-r` *access_order*	Specifies dimension access order. *(Default: `1,2`)* *access_order* is a comma-separated list of integers specifying the order of access. For example, `-r 1,3,2` specifies the traversal of dimension 1 first, then dimension 3, and finally dimension 2.
`-t`	Selects extendable HDF5 dataset dimensions. *(Default: Datasets are fixed size.)*
`-v` *file_driver*	Selects HDF5 driver to be used for HDF5 file access. *(Default: `sec2`)* Valid values are as follows: &nbsp `sec2` &nbsp `stdio` &nbsp `core` &nbsp `split` &nbsp `multi` &nbsp `family` &nbsp `direct`
`-w`	Specifies the performance of write tests only, read performance will not be tested. *(Default: Both write and read tests are performed.)*
`-x` *buffer_size_list*	Specifies transfer buffer dimensions and sizes. *(Default: `10,20`)*

History:

Release	**Command Line Tool**
1.8.1	Tool introduced in this release.

Tool Name: h5redeploy

Syntax:

`h5redeploy` [`help` | `-help`]

`h5redeploy` [`-echo`] [`-force`] [`-prefix=`*dir*] [`-tool=`*tool*] [`-show`]

Purpose:

Updates HDF5 compiler tools after an HDF5 software installation in a new location.

Description:

`h5redeploy` updates the HDF5 compiler tools after the HDF5 software has been installed in a new location.

Options and Parameters:

`help`, `-help`	Prints a help message.
`-echo`	Shows all the shell commands executed.
`-force`	Performs the requested action without offering any prompt requesting confirmation.
`-prefix=`*dir*	Specifies a new directory in which to find the HDF5 subdirectories `lib/` and `include/`. (Default: current working directory)
`-tool=`*tool*	Specifies the tool to update. *tool* must be in the current directory and must be writable. (Default: `h5cc`)
`-show`	Shows all of the shell commands to be executed without actually executing them.

History:

Release	**Command Line Tool**
1.6.0	Tool introduced in this release.

Tool Name: h5cc and h5pcc

Syntax:

`h5cc [`*OPTIONS*`]` *<compile line>*

`h5pcc [`*OPTIONS*`]` *<compile_line>*

Purpose:

Helper scripts to compile HDF5 applications.

Description:

`h5cc` and `h5pcc` can be used in much the same way as `mpicc` by MPICH is used to compile an HDF5 program. These tools take care of specifying on the command line the locations of the HDF5 header files and libraries. `h5cc` is for use in serial computing environments; `h5pcc` is for parallel environments.

`h5cc` and `h5pcc` subsume all other compiler scripts in that if you have used a set of scripts to compile the HDF5 library, then `h5cc` and `h5pcc` also use those scripts. For example, when compiling an MPICH program, you use the `mpicc` script. If you have built HDF5 using MPICH, then `h5cc` uses the MPICH program for compilation.

Some programs use HDF5 in only a few modules. It is not necessary to use `h5cc` or `h5pcc` to compile those modules which do not use HDF5. In fact, since `h5cc` and `h5pcc` are only convenience scripts, you can still compile HDF5 modules in the normal manner, though you will have to specify the HDF5 libraries and include paths yourself. Use the `-show` option to see the details.

An example of how to use `h5cc` to compile the program `hdf_prog`, which consists of the modules `prog1.c` and `prog2.c` and uses the HDF5 shared library, would be as follows. `h5pcc` is used in an identical manner.

```
        # h5cc -c prog1.c
        # h5cc -c prog2.c
        # h5cc -shlib -o hdf_prog prog1.o prog2.o
```

Options and Parameters:

`-help`	Prints a help message.
`-echo`	Show all the shell commands executed.
`-prefix=DIR`	Use the directory *`DIR`* to find the HDF5 `lib/` and `include/` subdirectories. Default: prefix specified when configuring HDF5.
`-show`	Show the commands without executing them.
`-shlib`	Compile using shared HDF5 libraries.
`-noshlib`	Compile using static HDF5 libraries [default].
<compile line>	The normal compile line options for your compiler. `h5cc` and `h5pcc` use the the same compiler you used to compile HDF5. Check your compiler's manual for more information on which options are needed.

Environment Variables:

When set, these environment variables override some of the built-in `h5cc` and `h5pcc` defaults.

`HDF5_CC`	Use a different C compiler.
`HDF5_CLINKER`	Use a different linker.
`HDF5_USE_SHLIB=[yes|no]`	Use shared version of the HDF5 library [default: no].

Tool Name: h5fc and h5pfc

Syntax:

`h5fc [`*OPTIONS*`]` *<compile line>*
`h5pfc [`*OPTIONS*`]` *<compile_line>*

Purpose:

Helper scripts to compile HDF5 Fortran90 applications.

Description:

`h5fc` and `h5pfc` can be used in much the same way `mpif90` by MPICH is used to compile an HDF5 program. These tools take care of specifying on the command line the locations of the HDF5 header files and libraries. `h5fc` is for use in serial computing environments; `h5pfc` is for parallel environments.

`h5fc` and `h5pfc` subsume all other compiler scripts in that if you have used a set of scripts to compile the HDF5 Fortran library, then `h5fc` and `h5pfc` also use those scripts. For example, when compiling an MPICH program, you use the `mpif90` script. If you have built HDF5 using MPICH, then `h5fc` uses the MPICH program for compilation.

Some programs use HDF5 in only a few modules. It is not necessary to use `h5fc` and `h5pfc` to compile those modules which do not use HDF5. In fact, since `h5fc` and `h5pfc` are only convenience scripts, you can still compile HDF5 Fortran modules in the normal manner, though you will have to specify the HDF5 libraries and include paths yourself. Use the `-show` option to see the details.

An example of how to use `h5fc` to compile the program `hdf_prog`, which consists of the modules `prog1.f90` and `prog2.f90` and uses the HDF5 Fortran library, would be as follows. `h5pfc` is used in an identical manner.

```
# h5fc -c prog1.f90
# h5fc -c prog2.f90
# h5fc -o hdf_prog prog1.o prog2.o
```

Options and Parameters:

`-help`	Prints a help message.
`-echo`	Show all the shell commands executed.
`-prefix=DIR`	Use the directory `DIR` to find HDF5 `lib/` and `include/` subdirectories Default: prefix specified when configuring HDF5.
`-show`	Show the commands without executing them.
<compile line>	The normal compile line options for your compiler. `h5fc` and `h5pfc` use the the same compiler you used to compile HDF5. Check your compiler's manual for more information on which options are needed.

Environment Variables:

When set, these environment variables override some of the built-in `h5fc` and `h5pfc` defaults.

`HDF5_FC`	Use a different Fortran90 compiler.
`HDF5_FLINKER`	Use a different linker.

History:

Release	**Command Line Tool**
1.6.0	Tool introduced in this release.

Tool Name: h5c++

Syntax:

`h5c++ [`*OPTIONS*`]` *<compile line>*

Purpose:

Helper script to compile HDF5 C++ applications.

Description:

`h5c++` can be used in much the same way MPIch is used to compile an HDF5 program. It takes care of specifying where the HDF5 header files and libraries are on the command line.

`h5c++` supersedes all other compiler scripts in that if you've used one set of compiler scripts to compile the HDF5 C++ library, then `h5c++` uses those same scripts. For example, when compiling an MPIch program, you use the `mpiCC` script.

Some programs use HDF5 in only a few modules. It isn't necessary to use `h5c++` to compile those modules which don't use HDF5. In fact, since `h5c++` is only a convenience script, you are still able to compile HDF5 C++ modules in the normal way. In that case, you will have to specify the HDF5 libraries and include paths yourself.

An example of how to use `h5c++` to compile the program `hdf_prog`, which consists of modules `prog1.cpp` and `prog2.cpp` and uses the HDF5 C++ library, would be as follows:

```
# h5c++ -c prog1.cpp
# h5c++ -c prog2.cpp
# h5c++ -o hdf_prog prog1.o prog2.o
```

Options and Parameters:

`-help`	Prints a help message.
`-echo`	Show all the shell commands executed.
`-prefix=DIR`	Use the directory `DIR` to find HDF5 `lib/` and `include/` subdirectories Default: prefix specified when configuring HDF5.
`-show`	Show the commands without executing them.
<compile line>	The normal compile line options for your compiler. `h5c++` uses the same compiler you used to compile HDF5. Check your compiler's manual for more information on which options are needed.

Environment Variables:

When set, these environment variables override some of the built-in defaults of `h5c++`.

`HDF5_CXX`	Use a different C++ compiler.
`HDF5_CXXLINKER`	Use a different linker.

History:

Release	**Command Line Tool**
1.6.0	Tool introduced in this release.

HDF5 Predefined Datatypes

The following datatypes are predefined in HDF5.

IEEE floating point datatypes

- 32-bit and 64-bit
- Big-endian and little-endian

```
H5T_IEEE_F32BE
H5T_IEEE_F32LE
H5T_IEEE_F64BE
H5T_IEEE_F64LE
```

Standard datatypes

- Signed integer (2's complement), unsigned integer, and bitfield
- 8-bit, 16-bit, 32-bit, and 64-bit
- Big-endian and little-endian

H5T_STD_I8BE	H5T_STD_U8BE	H5T_STD_B8BE
H5T_STD_I8LE	H5T_STD_U8LE	H5T_STD_B8LE
H5T_STD_I16BE	H5T_STD_U16BE	H5T_STD_B16BE
H5T_STD_I16LE	H5T_STD_U16LE	H5T_STD_B16LE
H5T_STD_I32BE	H5T_STD_U32BE	H5T_STD_B32BE
H5T_STD_I32LE	H5T_STD_U32LE	H5T_STD_B32LE
H5T_STD_I64BE	H5T_STD_U64BE	H5T_STD_B64BE
H5T_STD_I64LE	H5T_STD_U64LE	H5T_STD_B64LE

- Object reference or dataset region reference

```
H5T_STD_REF_OBJ
H5T_STD_REF_DSETREG
```

UNIX-specific datatypes

- 32-bit and 64-bit
- Big-endian and little-endian

```
H5T_UNIX_D32BE
H5T_UNIX_D32LE
H5T_UNIX_D64BE
H5T_UNIX_D64LE
```

C-specific datatype

- String datatype in C (size defined in bytes rather than in bits)

```
H5T_C_S1
```

FORTRAN-specific datatype

- String datatype in FORTRAN (as defined for the HDF5 C library)

```
H5T_FORTRAN_S1
```

Intel-specific datatypes

- For Intel CPUs
- Little-endian
- Signed integer (2's complement), unsigned integer, bitfield, and IEEE floating point
- 8-bit, 16-bit, 32-bit, and 64-bit

```
H5T_INTEL_I8                         H5T_INTEL_B8
H5T_INTEL_I16                        H5T_INTEL_B16
H5T_INTEL_I32                        H5T_INTEL_B32
H5T_INTEL_I64                        H5T_INTEL_B64

H5T_INTEL_U8                         H5T_INTEL_F32
H5T_INTEL_U16                        H5T_INTEL_F64
H5T_INTEL_U32
H5T_INTEL_U64
```

DEC Alpha-specific datatypes

- For DEC Alpha CPUs
- Little-endian
- Signed integer (2's complement), unsigned integer, bitfield, and IEEE floating point
- 8-bit, 16-bit, 32-bit, and 64-bit

```
H5T_ALPHA_I8                         H5T_ALPHA_B8
H5T_ALPHA_I16                        H5T_ALPHA_B16
H5T_ALPHA_I32                        H5T_ALPHA_B32
H5T_ALPHA_I64                        H5T_ALPHA_B64

H5T_ALPHA_U8                         H5T_ALPHA_F32
H5T_ALPHA_U16                        H5T_ALPHA_F64
H5T_ALPHA_U32
H5T_ALPHA_U64
```

OpenVMS DEC Alpha-specific datatypes

- For OpenVMS on DEC Alpha CPUs
- VAX byte order
- 32- and 64-bit floating point

```
H5T_VAX_F32 (Corresponds to F_Floating type)
H5T_VAX_F64 (Corresponds to G_Floating type)
```

MIPS-specific datatypes

- For MIPS CPUs, commonly used in SGI system
- Big-endian
- Signed integer (2's complement), unsigned integer, bitfield, and IEEE floating point
- 8-bit, 16-bit, 32-bit, and 64-bit

```
H5T_MIPS_I8                          H5T_MIPS_B8
H5T_MIPS_I16                         H5T_MIPS_B16
H5T_MIPS_I32                         H5T_MIPS_B32
H5T_MIPS_I64                         H5T_MIPS_B64

H5T_MIPS_U8                          H5T_MIPS_F32
H5T_MIPS_U16                         H5T_MIPS_F64
H5T_MIPS_U32
H5T_MIPS_U64
```

Predefined native datatypes

These are the datatypes detected by `H5detect`. Their names differ from other HDF5 datatype names as follows:

- ♦ Instead of a class name, precision, and byte order as the last component, they have a C-like datatype name.
- ♦ If the datatype begins with `U`, then it is the unsigned version of the integer datatype; other integer datatypes are signed.
- ♦ The datatype `LLONG` corresponds to C's `long_long` and `LDOUBLE` is `long_double`. These datatypes might be the same as `LONG` and `DOUBLE`, respectively.

```
H5T_NATIVE_CHAR                      H5T_NATIVE_FLOAT
H5T_NATIVE_SCHAR                     H5T_NATIVE_DOUBLE
H5T_NATIVE_UCHAR                     H5T_NATIVE_LDOUBLE

H5T_NATIVE_SHORT                     H5T_NATIVE_B8
H5T_NATIVE_USHORT                    H5T_NATIVE_B16
                                     H5T_NATIVE_B32
H5T_NATIVE_INT                       H5T_NATIVE_B64
H5T_NATIVE_UINT
                                     H5T_NATIVE_OPAQUE
H5T_NATIVE_LONG                      H5T_NATIVE_HADDR
H5T_NATIVE_ULONG                     H5T_NATIVE_HSIZE
H5T_NATIVE_LLONG                     H5T_NATIVE_HSSIZE
H5T_NATIVE_ULLONG                    H5T_NATIVE_HERR
                                     H5T_NATIVE_HBOOL
```

ANSI C9x-specific native integer datatypes

- Signed integer (2's complement), unsigned integer, and bitfield
- 8-bit, 16-bit, 32-bit, and 64-bit
- LEAST -- storage to use least amount of space
  FAST -- storage to maximize performance

```
H5T_NATIVE_INT8                     H5T_NATIVE_INT32
H5T_NATIVE_UINT8                    H5T_NATIVE_UINT32
H5T_NATIVE_INT_LEAST8               H5T_NATIVE_INT_LEAST32
H5T_NATIVE_UINT_LEAST8              H5T_NATIVE_UINT_LEAST32
H5T_NATIVE_INT_FAST8                H5T_NATIVE_INT_FAST32
H5T_NATIVE_UINT_FAST8               H5T_NATIVE_UINT_FAST32

H5T_NATIVE_INT16                    H5T_NATIVE_INT64
H5T_NATIVE_UINT16                   H5T_NATIVE_UINT64
H5T_NATIVE_INT_LEAST16              H5T_NATIVE_INT_LEAST64
H5T_NATIVE_UINT_LEAST16             H5T_NATIVE_UINT_LEAST64
H5T_NATIVE_INT_FAST16               H5T_NATIVE_INT_FAST64
H5T_NATIVE_UINT_FAST16              H5T_NATIVE_UINT_FAST64
```

FORTRAN90 API datatypes

- Datatypes defined for the FORTRAN90 APIs

- Native integer, single-precision real, double-precision real, and character

```
H5T_NATIVE_INTEGER
H5T_NATIVE_REAL
H5T_NATIVE_DOUBLE
H5T_NATIVE_CHARACTER
```

- Signed integer (2's complement), unsigned integer, and IEEE floating point
- 8-bit, 16-bit, 32-bit, and 64-bit
- Big-endian and little-endian

```
H5T_STD_I8BE            H5T_STD_U8BE            H5T_IEEE_F32BE
H5T_STD_I8LE            H5T_STD_U8LE            H5T_IEEE_F32LE
H5T_STD_I16BE           H5T_STD_U16BE           H5T_IEEE_F64BE
H5T_STD_I16LE           H5T_STD_U16LE           H5T_IEEE_F64LE
H5T_STD_I32BE           H5T_STD_U32BE
H5T_STD_I32LE           H5T_STD_U32LE
H5T_STD_I64BE           H5T_STD_U64BE
H5T_STD_I64LE           H5T_STD_U64LE
```

- Object reference or dataset region reference

```
H5T_STD_REF_OBJ
H5T_STD_REF_DSETREG
```

HDF5 Fortran90 Flags, Datatypes and User’s Notes

Fortran90 Datatypes

The Fortran90 HDF5 datatypes are listed in HDF5 Predefined Datatypes

Fortran90 Flags

The Fortran90 HDF5 flags have the same meanings as the C flags defined in the *HDF5 Reference Manual* and the *HDF5 User's Guide*.

File access flags

H5F_ACC_RDWR_F	H5F_ACC_EXCL_F	H5F_SCOPE_LOCAL_F
H5F_ACC_RDONLY_F	H5F_ACC_DEBUG_F	H5F_SCOPE_GLOBAL_F
H5F_ACC_TRUNC_F		

Group management flags

H5G_UNKNOWN_F	H5G_DATASET_F	H5G_LINK_HARD_F
H5G_LINK_F	H5G_TYPE_F	H5G_LINK_SOFT_F
H5G_GROUP_F	H5G_LINK_ERROR_F	

Dataset format flags

H5D_COMPACT_F	H5D_CONTIGUOUS_F	H5D_CHUNKED_F

MPI IO data transfer flags

H5FD_MPIO_INDEPENDENT_F	H5FD_MPIO_COLLECTIVE_F

Error flags

H5E_NONE_MAJOR_F	H5E_CACHE_F	H5E_STORAGE_F
H5E_ARGS_F	H5E_BTREE_F	H5E_PLIST_F
H5E_RESOURCE_F	H5E_SYM_F	H5E_ATTR_F
H5E_INTERNAL_F	H5E_HEAP_F	H5E_PLINE_F
H5E_FILE_F	H5E_OHDR_F	H5E_EFL_F
H5E_IO_F	H5E_DATATYPE_F	H5E_REFERENCE_F
H5E_FUNC_F	H5E_DATASPACE_F	H5E_VFL_F
H5E_ATOM_F	H5E_DATASET_F	H5E_TBBT_F

Object identifier flags

H5I_FILE_F	H5I_DATASPACE_F	H5I_BADID_F
H5I_GROUP_F	H5I_DATASET_F	
H5I_DATATYPE_F	H5I_ATTR_F	

Property list flags

```
H5P_FILE_CREATE_F        H5P_DATASET_CREATE_F       H5P_MOUNT_F
H5P_FILE_ACCESS_F        H5P_DATASET_XFER_F         H5P_DEFAULT_F
```

Reference pointer flags

```
H5R_OBJECT_F          H5R_DATASET_REGION_F
```

Dataspace flags

```
H5S_SCALAR_F           H5S_SELECT_SET_F        H5S_UNLIMITED_F
H5S_SIMPLE_F           H5S_SELECT_OR_F         H5S_ALL_F
```

Datatype flags

```
H5T_NO_CLASS_F       H5T_ORDER_LE_F            H5T_NORM_IMPLIED_F
H5T_INTEGER_F        H5T_ORDER_BE_F            H5T_NORM_MSBSET_F
H5T_FLOAT_F          H5T_ORDER_VAX_F           H5T_NORM_NONE_F
H5T_TIME_F           H5T_PAD_ZERO_F            H5T_CSET_ASCII_F
H5T_STRING_F         H5T_PAD_ONE_F             H5T_STR_NULLTERM_F
H5T_BITFIELD_F       H5T_PAD_BACKGROUND_F      H5T_STR_NULLPAD_F
H5T_OPAQUE_F         H5T_PAD_ERROR_F           H5T_STR_SPACEPAD_F
H5T_COMPOUND_F       H5T_SGN_NONE_F            H5T_STR_ERROR_F
H5T_REFERENCE_F      H5T_SGN_2_F
H5T_ENUM_F           H5T_SGN_ERROR_F
```

HDF5 Fortran90 User's Notes

About the source code organization

The Fortran APIs are organized in modules parallel to the HDF5 Interfaces. Each module is in a separate file with the name `H5*ff.f`. Corresponding C stubs are in the `H5*f.c` files. For example, the Fortran File APIs are in the file `H5Fff.f` and the corresponding C stubs are in the file `H5Ff.c`.

Each module contains Fortran definitions of the constants, interfaces to the subroutines if needed, and the subroutines themselves.

Users must use constant names in their programs instead of the numerical values, as the numerical values are subject to change without notice.

About the Fortran APIs

The Fortran APIs come in the form of Fortran subroutines with the following characteristics:

- Each Fortran subroutine name is derived from the corresponding C function name by adding `"_f"` to the name. For example, the name of the C function to create an HDF5 file is `H5Fcreate`; the corresponding Fortran subroutine is `h5fcreate_f`.
- A description of each implemented Fortran subroutine and its parameters can be found following the description of the corresponding C function in the HDF5 Reference Manual provided with this release.
- The parameter list for each Fortran subroutine has two more parameters than the corresponding C function. These additional parameters hold the return value and an error code. The order of the Fortran

subroutine parameters may differ from the order of the C function parameters. The Fortran subroutine parameters are listed in the following order:

- ♦ Required input parameters
- ♦ Output parameters, including return value and error code
- ♦ Optional input parameters

For example, the C function to create a dataset has the following prototype:

```
hid_t H5Dcreate(hid_it loc_id, char *name, hid_t type_id,
      hid_t space_id, hid_t creation_prp);
```

The corresponding Fortran subroutine has the following form:

```
SUBROUTINE h5dcreate_f(loc_id, name, type_id, space_id, dset_id,
      hdferr, creation_prp)
```

The first four parameters of the Fortran subroutine correspond to the C function parameters. The fifth parameter, `dset_id`, is an output parameter and contains a valid dataset identifier if the value of the sixth output parameter hdferr indicates successful completion. (Error code descriptions are provided with the subroutine descriptions in the Reference Manual.) The seventh input parameter, `creation_prp`, is optional, and may be omitted when the default creation property list is used.

- Parameters to the Fortran subroutines have one of the following predefined datatypes (see the file `H5fortran_types.f90` for `KIND` definitions):
  - ♦ *INTEGER(HID_T)* compares with the `hid_t` datatype in the HDF5 C APIs.
  - ♦ `INTEGER(HSIZE_T)` compares with `hsize_t` in the HDF5 C APIs.
  - ♦ `INTEGER(HSSIZE_T)` compares with `hssize_t` in the HDF5 C APIs.
  - ♦ `INTEGER(SIZE_T)` compares with the C `size_t` datatype.

  These integer types usually correspond to 4 or 8 byte integers, depending on the FORTRAN90 compiler and the corresponding HDF5 C library definitions.

  The H5R module defines two types of references:
  - ♦ `TYPE(HOBJ_REF_T_F)` compares to `hobj_ref_t` in the HDF5 C API.
  - ♦ `TYPE(HDSET_REG_REF_T_F)` compares to `hdset_reg_ref_t` in the HDF5 C API.

- Each Fortran application must call the `h5open_f` subroutine to initialize the Fortran interface and the HDF5 C Library before calling any HDF5 Fortran subroutine. The application must call the `h5close_f` subroutine after all calls to the HDF5 Fortran Library to close the Fortran interface and HDF5 C Library.

- List of the predefined datatypes can be found in the HDF5 Reference Manual provided with this release. See HDF5 Predefined Datatypes.

- When a C application reads data stored from a Fortran program, the data will appear to be transposed due to the difference in the C and Fortran storage orders. For example, if Fortran writes a 4x6 two-dimensional dataset to the file, a C program will read it as a 6x4 two-dimensional dataset into memory. The HDF5 C utilities h5dump and h5ls will also display transposed data, if data is written from a Fortran program.

- Fortran indices are 1-based.

- Compound datatype datasets can be written or read by atomic fields only.

API Compatibility Macros in HDF5

Audience

The target audience for this document has existing applications that use the HDF5 Library, and is considering moving to HDF5 Release 1.8.0 to take advantage of the latest library features and enhancements.

Compatibility Issues

HDF5 1.8.0 is a major update of the HDF5 Library. Several compatibility issues must be considered when migrating applications to the HDF5 1.8.0 release.

This document, *API Compatibility Macros in HDF5*, introduces the approach taken by The HDF Group (THG) in HDF Release 1.8.0 to help existing users of HDF5 address compatibility issues in the HDF5 API. The companion document, *New Features in HDF5 Release 1.8.0 and Format Compatibility Considerations*, discusses features introduced in HDF5 Release 1.8.0, the HDF5 API calls associated with those features, and the potential file format compatibility issues that may result if the new features are used.

Summary and Motivation

In response to new and evolving requirements for the library and data format, several basic functions have changed since HDF5 was first released. To allow existing applications to continue to compile and run properly, all versions of these functions have been retained in the later releases of the HDF5 Library.

HDF5 Release 1.8.0 includes a number of new features that will offer many users of HDF5 substantial performance improvements and expanded capabilities. Many of these features can only be accessed via revised API calls. Given the scope of the changes, and recognizing the potentially time-consuming task of editing all the affected calls in user applications, THG has created a set of macros that can be used to flexibly and easily map existing API calls to either 1.6.x or 1.8.x (currently 1.8.0) functions. We refer to these as the *API Compatibility Macros*.

THG generally encourages users to update applications to work with the latest HDF5 library release, so that all new features and enhancements are available to them. At the same time, THG understands that under some circumstances updating applications may not be feasible or necessary. The API Compatibility Macros, described in this document, provide a bridge from old APIs to new, and can be particularly helpful in situations such as these:

- Source code is not available - only binaries are available; updating the application is not feasible.
- Source code is available, but there are no resources to update it.
- Source code is available, as are resources to update it, but the old version works quite well so updates are not a priority. At the same time, it is desirable to take advantage of certain efficiencies in the newer HDF5 release that do not require code changes.
- Source code is available, as are resources to update it, but the applications are large or complex, and must continue to run while the code updates are carried out.

Understanding and Using the Macros

As part of HDF5 release 1.8.0, twenty-three functions that existed in previous versions of the library were updated with new calling parameters and given new names. The updated versions of the functions have a "2" at the end of the original function name. The original versions of these functions were retained and renamed to have a "1" at the end of the original function name. API Compatibility Macros, with the same names as the original function names, were created.

Concretely, consider the function `H5Acreate` in HDF5 releases prior to 1.8.0:

Original function name and signature, in releases prior to 1.8.0:
`hid_t H5Acreate( hid_t loc_id, const char *attr_name, hid_t type_id, hid_t space_id, hid_t acpl_id )`

Updated function and signature, introduced in release 1.8.0:
`hid_t H5Acreate2( hid_t loc_id, const char *attr_name, hid_t type_id, hid_t space_id, hid_t acpl_id, hid_t aapl_id )`

Original function and signature, renamed in release 1.8.0:
`hid_t H5Acreate1( hid_t loc_id, const char *attr_name, hid_t type_id, hid_t space_id, hid_t acpl_id )`

API Compatibility Macro, introduced in release 1.8.0:
`H5Acreate` The macro, `H5Acreate`, will be mapped to either `H5Acreate1` or `H5Acreate2`. The mapping is determined by a combination of the configuration options use to build the HDF5 Library and compile-time options used to build the application. The calling parameters used with the `H5Acreate` Compatibility Macro should match the number and type of the function they will be mapped to (`H5Acreate1` or `H5Acreate2`).

The function names ending in "1" or "2" are referred to as *version-numbered names*, and the corresponding functions are referred to as *version-numbered functions*. For new code development, THG recommends use of the Compatibility Macro mapped to the latest version of the function. The original versions of these functions, with names ending in "1", should be considered deprecated and, in general, should not be used when developing new code.

Compatibility Macro Mapping Options

To determine the mapping for a given API Compatibility Macro in a given application, a combination of user-controlled selections, collectively referred to as the *Compatibility Macro Mapping Options*, is considered in the following sequence:

1. What Compatibility Macro configuration option was used to build the HDF5 Library? We refer to this selection as the *Library Mapping*.
2. Was a Compatibility Macro global compile-time option specified when the application was built? We refer to this (optional) selection as the *Application Mapping*. If an Application Mapping exists, it overrides the Library Mapping.
3. Were any Compatibility Macro function-level compile-time options specified when the application was built? We refer to these (optional) selections as *Function Mappings*. If Function Mappings exist, they override Library and Application Mappings for the relevant API Compatibility Macros.

The tables that follow summarize the macro mapping behaviors, and the configuration and compile-time options that control the mappings. The macro `H5Gcreate` is used to demonstrate mapping behavior.

Regardless of the Macro Mapping Options used, the 1.8.x functions will always be available by explicitly calling the version-numbered functions by their version-numbered names. For example, `H5Gcreate2`. Through the Compatibility Macro Mapping Options provided, it is possible to disallow calls to the deprecated 1.6.x functions, such as `H5Gcreate1`. This capability can be used to guarantee only the most recent versions of the functions are being called.

Library Mapping Options

When the HDF5 Library is built, `configure` flags can be used to control the API Compatibility Macro Mapping behavior exhibited by the library. This behavior can be overridden by Application and Function Mappings. One configure flag excludes deprecated functions from the HDF5 library, making them unavailable to applications linked with the library.

Table 1: Library Mapping Options

`configure` flag	**Macros map to Release (version-numbered function; `H5Gcreate` shown)**	**deprecated functions available? (`H5Gcreate1`)**
`--with-default-api-version=v18` *Also, default behavior if no flag specified.*	1.8.x (`H5Gcreate2`)	yes
`--with-default-api-version=v16`	1.6.x (`H5Gcreate1`)	yes
`--disable-deprecated-symbols`	1.8.x (`H5Gcreate2`)	no

Refer to the file `libhdf5.settings` in the directory where the HDF5 library is installed to determine the `configure` flags used to build the library. In particular, look for the two lines shown here:

```
Default Version of Public Symbols: v18
With Deprecated Public Symbols: Yes
```

Application Mapping Options

When an application using HDF5 APIs is built and linked with the HDF5 Library, compile-time options to `h5cc` can be used to control the API Compatibility Macro Mapping behavior exhibited by the application. The Application Mapping overrides the behavior specified by the Library Mapping, and can be overridden on a function-by-function basis by the Function Mappings.

If the HDF5 Library was configured with the `--disable-deprecated-symbols` flag, then the deprecated functions will not be available, regardless of the Application Mapping options.

Table 2: Application Mapping Options

`h5cc` option	**Macros map to Release (version-numbered function; `H5Gcreate` shown)**	**deprecated functions available? (`H5Gcreate1`)**
Default behavior if no option specified.	1.8.x (`H5Gcreate2`)	yes* **if available in library*
`-D H5_USE_16_API`	1.6.x (`H5Gcreate1`)	yes* **if available in library*
`-D H5_NO_DEPRECATED_SYMBOLS`	1.8.x (`H5Gcreate2`)	no

Function Mapping Options

Function Mappings are specified when the application is built. These mappings can be used to control the mapping of the API Compatibility Macros to underlying functions on a function-by-function basis. The Function Mappings override the Library and Application Mappings discussed earlier.

If the HDF5 Library was configured with the `--disable-deprecated-symbols` flag, or `-D H5_NO_DEPRECATED_SYMBOLS` is used to compile the application, then the deprecated functions will not be available, regardless of the Function Mapping options.

For every function with multiple available versions, a compile-time version flag can be defined to selectively map the function macro to the desired version-numbered function. For example, the `H5Gcreate` can be mapped to either `H5Gcreate1` or `H5Gcreate2`. When used, the value of the `H5Gcreate_vers` compile-time version flag determines which function will be called:

- When `H5Gcreate_vers` is set to `1`, the macro `H5Gcreate` will be mapped to `H5Gcreate1`.
  ```
  h5cc ... -D H5Gcreate_vers=1 ...
  ```
- When `H5Gcreate_vers` is set to `2`, the macro `H5Gcreate` will be mapped to `H5Gcreate2`.
  ```
  h5cc ... -D H5Gcreate_vers=2 ...
  ```
- When `H5Gcreate_vers` is not set, the macro `H5Gcreate` will be mapped to either `H5Gcreate1` or `H5Gcreate2`, based on the Application Mapping, if one was specified, or on the Library Mapping.
  ```
  h5cc ...
  ```

As of Release 1.8.0, the API Compatibility Macros, the Function Mapping compile-time version flags and values, and the corresponding version-numbered functions are as indicated:

Table 3: Function Mapping Options

Macro	`h5cc` version flag and value	Mapped To Function
`H5Acreate`	`-D H5Acreate_vers=1`	`H5Acreate1`
	`-D H5Acreate_vers=2`	`H5Acreate2`
`H5Adelete`	`-D H5Adelete_vers=1`	`H5Adelete1`
	`-D H5Adelete_vers=2`	`H5Adelete2`
`H5Aiterate`	`-D H5Aiterate_vers=1`	`H5Aiterate1`
	`-D H5Aiterate_vers=2`	`H5Aiterate2`
`H5Arename`	`-D H5Arename_vers=1`	`H5Arename1`
	`-D H5Arename_vers=2`	`H5Arename2`
`H5Dcreate`	`-D H5Dcreate_vers=1`	`H5Dcreate1`
	`-D H5Dcreate_vers=2`	`H5Dcreate2`
`H5Dopen`	`-D H5Dopen_vers=1`	`H5Dopen1`
	`-D H5Dopen_vers=2`	`H5Dopen2`
`H5Eclear`	`-D H5Eclear_vers=1`	`H5Eclear1`
	`-D H5Eclear_vers=2`	`H5Eclear2`
`H5Eprint`	`-D H5Eprint_vers=1`	`H5Eprint1`
	`-D H5Eprint_vers=2`	`H5Eprint2`
`H5Epush`	`-D H5Epush_vers=1`	`H5Epush1`
	`-D H5Epush_vers=2`	`H5Epush2`
`H5Eset_auto`	`-D H5Eset_auto_vers=1`	`H5Eset_auto1`
	`-D H5Eset_auto_vers=2`	`H5Eset_auto2`
`H5Eget_auto`	`-D H5Eget_auto_vers=1`	`H5Eget_auto1`
	`-D H5Eget_auto_vers=2`	`H5Eget_auto2`
`H5Ewalk`	`-D H5Ewalk_vers=1`	`H5Ewalk1`
	`-D H5Ewalk_vers=2`	`H5Ewalk2`
`H5Gcreate`	`-D H5Gcreate_vers=1`	`H5Gcreate1`
	`-D H5Gcreate_vers=2`	`H5Gcreate2`
`H5Gopen`	`-D H5Gopen_vers=1`	`H5Gopen1`
	`-D H5Gopen_vers=2`	`H5Gopen2`
`H5Pget_filter`	`-D H5Pget_filter_vers=1`	`H5Pget_filter1`
	`-D H5Pget_filter_vers=2`	`H5Pget_filter2`

H5Pget_filter_by_id	-D H5Pget_filter_by_id_vers=1	H5Pget_filter_by_id1
	-D H5Pget_filter_by_id_vers=2	H5Pget_filter_by_id2
H5Pinsert	-D H5Pinsert_vers=1	H5Pinsert1
	-D H5Pinsert_vers=2	H5Pinsert2
H5Pregister	-D H5Pregister_vers=1	H5Pregister1
	-D H5Pregister_vers=2	H5Pregister2
H5Rget_obj_type	-D H5Rget_obj_typevers=1	H5Rget_obj_type1
	-D H5Rget_obj_type_vers=2	H5Rget_obj_type2
H5Tarray_create	-D H5Tarray_create_vers=1	H5Tarray_create1
	-D H5Tarray_create_vers=2	H5Tarray_create2
H5Tcommit	-D H5Tcommit_vers=1	H5Tcommit1
	-D H5Tcommit_vers=2	H5Tcommit2
H5Tget_array_dims	-D H5Tget_array_dims_vers=1	H5Tget_array_dims1
	-D H5Tget_array_dims_vers=2	H5Tget_array_dims2
H5Topen	-D H5Topen_vers=1	H5Topen1
	-D H5Topen_vers=2	H5Topen2

See the HDF5 Reference Manual for complete descriptions of all API Compatibility Macros and version-numbered functions shown in Table 3.

It is possible to specify multiple Function Mappings for a single application build:

`h5cc ... -D H5Gcreate_vers=1 -D H5Dcreate_vers=2...` As a result of the Function Mappings in this compile example, all occurrences of the macro H5Gcreate will be mapped to H5Gcreate1, and all occurrences of the macro H5Dcreate will be mapped to H5Dcreate2 for the application being built.

The Function Mappings can be used to guarantee that a given API Compatibility Macro will be mapped to the desired underlying function version regardless of the Library or Application Mappings. In cases where an application may benefit greatly from features offered by some of the later APIs, or must continue to use some earlier API versions for compatibility reasons, this fine-grained control may be very important.

As noted earlier, the Function Mappings can only reference version-numbered functions that are included in the HDF5 library, as determined by the configure flag used to build the library. For example, if the HDF5 library being linked with the application was built with the `--disable-deprecated-symbols` option, version 1 of the underlying functions would not be available, and the example above that defined `H5Gcreate_ver=1` would not be supported.

The Function Mappings do not negate any available functions. If `H5Gcreate1` is available in the installed version of the HDF5 Library, and the application was not compiled with the `-D H5_NO_DEPRECATED_SYMBOLS` flag, the function `H5Gcreate1` will remain available to the application through its version-numbered name. Similarly, `H5Gcreate2` will remain available to the application as `H5Gcreate2`. The Function Mapping version flag `H5Gcreate_vers` only controls the mapping of the API Compatibility Macro `H5GCreate` to one of the two available functions.

Compatibility Macros in HDF5 1.6.8 and Later

A series of similar compatibility macros have been introduced into the release 1.6 series of the library, starting with release 1.6.8. These macros simply alias the "1" version functions listed above, as well as the typedefs not listed, to their original non-numbered names.

This allows users to write code that can be used with any version of the library since 1.6.8 and any library compilation options except `H5_NO_DEPRECATED_SYMBOLS`, by always using the "1" version of versioned functions and types. For example, `H5Gcreate1` will always be interpreted in exactly the same manner by any version of the library since 1.6.8.

This can be especially useful in any case where the programmer does not have direct control over global macro definitions, such as when writing code meant to be copied to multiple applications or when writing code in a header file.

Common Use Case

A common scenario where the API Compatibility Macros may be helpful is the migration of an existing application to HDF5 Release 1.8.0. An incremental migration plan is outlined here:

1. Build the HDF5 library without specifying any Library Mapping `configure` flag. In this default mode, both 1.6.x and 1.8.x versions of the underlying functions are available, and the API Compatibility Macros will be mapped to the 1.8.x version-numbered functions. For example, `H5Gcreate` will be mapped to `H5Gcreate2`.
2. Compile the application with the `-D H5_USE_16_API` Application Mapping option, and link with the HDF5 library built in step 1. No changes should be required to build the application. The API Compatibility Macros, for example `H5Gcreate`, replace the actual function names that were used in versions of the library prior to 1.8.0. Because the Application Mapping overrides the Library Mapping, the macros will all be mapped to the 1.6.x versions of the functions.
3. Remap one API Compatibility Macro at a time (or sets of Macros), to use the 1.8.x versions. At each stage, use the Function Mappings to map the Macros being worked on to the 1.8.x versions. For example, use the `-D H5Gcreate_vers=2` version flag setting to remap the `H5Gcreate` macro to `H5Gcreate2`, the 1.8.x version. During this step, the application code will need to be modified to change the calling parameters used with the API Compatibility Macros to match the number and type of the 1.8.x version-numbered functions. The Macro name, for example `H5Gcreate`, should continue to be used in the code, to allow for possible re-mappings to later version-numbered functions in a future release.
4. After all Macros have been migrated to the 1.8.x version-numbered functions in step 3, compile the application without any Application or Function Mappings. This build uses the Library Mappings set in step 1, and maps API Compatibility Macros to the 1.8.x versions.
5. Finally, compile the application with the Application Mapping `-D H5_NO_DEPRECATED_SYMBOLS`, and address any failures to complete the application migration process.

HDF5 Glossary

---

atomic datatype
: A datatype which cannot be decomposed into smaller units at the API level.

attribute
: A small dataset that can be used to describe the nature and/or the intended usage of the object it is attached to.

chunked layout
: The storage layout of a chunked dataset.

chunking
: A storage layout where a dataset is partitioned into fixed-size multi-dimensional chunks. Chunking tends to improve performance and facilitates dataset extensibility.

compound datatype
: A collection of one or more atomic types or small arrays of such types. Similar to a struct in C or a common block in Fortran.

contiguous layout
: The storage layout of a dataset that is not chunked, so that the entire data portion of the dataset is stored in a single contiguous block.

data transfer property list
: The data transfer property list is used to control various aspects of the I/O, such as caching hints or collective I/O information.

dataset
: A multi-dimensional array of data elements, together with supporting metadata.

dataset access property list
A property list containing information on how a dataset is to be accessed.

dataset creation property list
A property list containing information on how raw data is organized on disk and how the raw data is compressed.

dataspace
An object that describes the dimensionality of the data array. A dataspace is either a regular N-dimensional array of data points, called a simple dataspace, or a more general collection of data points organized in another manner, called a complex dataspace.

datatype
An object that describes the storage format of the individual data points of a data set. There are two categories of datatypes: atomic and compound datatypes. An atomic type is a type which cannot be decomposed into smaller units at the API level. A compound datatype is a collection of one or more atomic types or small arrays of such types.

enumeration datatype
A one-to-one mapping between a set of symbols and a set of integer values, and an order is imposed on the symbols by their integer values. The symbols are passed between the application and library as character strings and all the values for a particular enumeration datatype are of the same integer type, which is not necessarily a native type.

file
A container for storing grouped collections of multi-dimensional arrays containing scientific data.

file access mode
Determines whether an existing file will be overwritten, opened for read-only access, or opened for read/write access. All newly created files are opened for both reading and writing.

file access property list
File access property lists are used to control different methods of performing I/O on files:

file creation property list
The property list used to control file metadata.

group
A structure containing zero or more HDF5 objects, together with supporting metadata. The two primary HDF5 objects are datasets and groups.

hard link
A direct association between a name and the object where both exist in a single HDF5 address space.

hyperslab
A portion of a dataset. A hyperslab selection can be a logically contiguous collection of points in a dataspace or a regular pattern of points or blocks in a dataspace.

identifier
A unique entity provided by the HDF5 library and used to access an HDF5 object, such as a file, goup, dataset, datatype, etc.

link
An association between a name and the object in an HDF5 file group.

member
A group or dataset that is in another dataset, *dataset A*, is a member of *dataset A*.

name
A slash-separated list of components that uniquely identifies an element of an HDF5 file. A name begins that begins with a slash is an absolute name which is accessed beginning with the root group of the file; all other names are relative names and the associated objects are accessed beginning with the current or specified group.

named datatype
A datatype that is named and stored in a file. Naming is permanent; a datatype cannot be changed after being named.

opaque datatype
A mechanism for describing data which cannot be otherwise described by HDF5. The only properties associated with opaque types are a size in bytes and an ASCII tag.

path
The slash-separated list of components that forms the name uniquely identifying an element of an HDF5 file.

property list
A collection of name/value pairs that can be passed to other HDF5 functions to control features that are typically unimportant or whose default values are usually used.

root group
The group that is the entry point to the group graph in an HDF5 file. Every HDF5 file has exactly one root group.

selection
(1) A subset of a dataset or a dataspace, up to the entire dataset or dataspace. (2) The elements of an array or dataset that are marked for I/O.

serialization
The flattening of an *N*-dimensional data object into a 1-dimensional object so that, for example, the data object can be transmitted over the network as a 1-dimensional bitstream.

soft link
An indirect association between a name and an object in an HDF5 file group.

storage layout
The manner in which a dataset is stored, either contiguous or chunked, in the HDF5 file.

super block
A block of data containing the information required to portably access HDF5 files on multiple platforms, followed by information about the groups and datasets in the file. The super block contains information about the size of offsets, lengths of objects, the number of entries in group tables, and additional version information for the file.

variable-length datatype
A sequence of an existing datatype (atomic, variable-length (VL), or compound) which are not fixed in length from one dataset location to another.