HDF5 Reference Manual

Preliminary Version for
Release 1.6.5
October, 2005

Hierarchical Data Format (HDF) Group
National Center for Supercomputing Applications (NCSA)
University of Illinois at Urbana–Champaign (UIUC)
Copyright Notice and Statement for
NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities

NCSA HDF5 (Hierarchical Data Format 5) Software Library and Utilities
All rights reserved.

Contributors: National Center for Supercomputing Applications (NCSA) at the University of Illinois at Urbana–Champaign (UIUC), Lawrence Livermore National Laboratory (LLNL), Sandia National Laboratories (SNL), Los Alamos National Laboratory (LANL), Jean-loup Gailly and Mark Adler (gzip library).

Redistribution and use in source and binary forms, with or without modification, are permitted for any purpose (including commercial purposes) provided that the following conditions are met:

1. Redistributions of source code must retain the above copyright notice, this list of conditions, and the following disclaimer.
2. Redistributions in binary form must reproduce the above copyright notice, this list of conditions, and the following disclaimer in the documentation and/or materials provided with the distribution.
3. In addition, redistributions of modified forms of the source or binary code must carry prominent notices stating that the original code was changed and the date of the change.
4. All publications or advertising materials mentioning features or use of this software are asked, but not required, to acknowledge that it was developed by the National Center for Supercomputing Applications at the University of Illinois at Urbana–Champaign and to credit the contributors.
5. Neither the name of the University nor the names of the Contributors may be used to endorse or promote products derived from this software without specific prior written permission from the University or the Contributors, as appropriate for the name(s) to be used.
6. THIS SOFTWARE IS PROVIDED BY THE UNIVERSITY AND THE CONTRIBUTORS "AS IS" WITH NO WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED. In no event shall the University or the Contributors be liable for any damages suffered by the users arising out of the use of this software, even if advised of the possibility of such damage.

Portions of HDF5 were developed with support from the University of California, Lawrence Livermore National Laboratory (UC LLNL). The following statement applies to those portions of the product and must be retained in any redistribution of source code, binaries, documentation, and/or accompanying materials:

This work was partially produced at the University of California, Lawrence Livermore National Laboratory (UC LLNL) under contract no. W–7405–ENG–48 (Contract 48) between the U.S. Department of Energy (DOE) and The Regents of the University of California (University) for the operation of UC LLNL.

DISCLAIMER: This work was prepared as an account of work sponsored by an agency of the United States Government. Neither the United States Government nor the University of California nor any of their employees, makes any warranty, express or implied, or assumes any liability or responsibility for the accuracy, completeness, or usefulness of any information, apparatus, product, or process disclosed, or represents that its use would not infringe privately– owned rights. Reference herein to any specific commercial products, process, or service by trade name, trademark, manufacturer, or otherwise, does not necessarily constitute or imply its endorsement, recommendation, or favoring by the United States Government or the University of California. The views and opinions of authors expressed herein do not necessarily state or reflect those of the United States Government or the University of California, and shall not be used for advertising or product endorsement purposes.
# Table of Contents

Overview 1  
Fortran90 and C++ APIs 2  
H5: General Library Functions 3  
H5A: Attribute Interface 13  
H5D: Datasets Interface 27  
H5E: Error Interface 49  
H5F: File Interface 61  
H5G: Group Interface 81  
H5I: Identifier Interface 101  
H5P: Property List Interface 109  
H5R: Reference Interface 245  
H5S: Dataspace Interface 253  
H5T: Datatype Interface 283  
H5Z: Compression Interface 351  
HDF5 Tools 361  
  h5dump 363  
  h5ls 368  
  h5diff 369  
  h5repack 371  
  h5perf 387  
  h5repart 373  
  h5redeploy 392  
  h5cc and h5pcc 393  
  h5fc and h5pfc 395  
  h5c++ 397  
Java–based tools (HDFview, etc.) *  
  h5import 374  
  gif2h5 382  
  h52gif 383  
  H4toH5 Conversion Library *  
  h5toh4 384  
  h4toh5 386  
HDF5 Predefined Datatypes 399  
HDF5 Glossary 403

* Links to descriptions of these tools appear on the HDF5 Tools page.
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.

See below for the FORTRAN90 and C++ APIs.

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

A PDF version of this HDF5 Reference Manual, formatted specifically for use as a printed book, is available at http://hdf.ncsa.uiuc.edu/HDF5/doc/PSandPDF/ within one week after each release.
The HDF5 Library distribution includes FORTRAN90 and C++ APIs, which are described in the following documents.

**Fortran90 API**

*HDF5 FORTRAN90 User's Notes* contains general information regarding the API. Specific information on each API call is found in the *HDF5 Reference Manual*.


Whereas Fortran subroutines had been described on separate pages in prior releases, those descriptions were fully integrated into the body of the reference manual for HDF5 Release 1.6.2 (and mostly so for Release 1.6.1).

*HDF5 Fortran90 Flags and Datatypes* lists the flags employed in the Fortran90 interface and contains a pointer to the HDF5 Fortran90 datatypes.

**C++ API**

*HDF5 C++ API Reference Manual*

This document supersedes all prior documentation of the C++ APIs.
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
- H5don't_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
- h5check_version_f
- h5get_libversion_f
- h5garbage_collect_f
- h5dont_atexit_f
- h5set_free_list_limits_f
**Name:** H5check_version

**Signature:**
```c
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:**
```fortran
SUBROUTINE h5check_version_f(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

```fortran
SUBROUTINE h5close_f(hdferr)
    IMPLICIT NONE
    INTEGER, INTENT(OUT) :: hdferr    ! Error code
END SUBROUTINE h5close_f
```
**Name:** H5dont_atexit  
**Signature:**  
\[ \text{herr_t H5dont_atexit(void)} \]  
**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

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 eliminate 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. There are three types of free lists:

◊ Regular free lists manage a single data structure.
◊ Array free lists manage arrays of a data structure.
◊ Block free lists manage blocks of bytes.

Alternate phrasing:

◊ Regular free lists manage data structures containing atomic data.
◊ Array free lists manage data structures containing array data.
◊ Block free lists manage blocks of bytes.

These are global limits, but each limit applies only to free lists of the specified type. Therefore, if an application sets a 1Mb limit on each of the global lists, up to 3Mb of total storage might be allocated, 1Mb for each of the regular, array, and block type lists.

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 limit on all regular free list memory used
- `int reg_list_lim`: IN: The limit on memory used in each regular free list
- `int arr_global_lim`: IN: The limit on all array free list memory used
- `int arr_list_lim`: IN: The limit on memory used in each array free list
- `int blk_global_lim`: IN: The limit on all block free list memory used
- `int blk_list_lim`: IN: The limit on memory used in each block 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.
H5A: Attribute Interface

Attribute API Functions

These functions create and manipulate attributes and information about attributes.

*The C Interfaces:*

- H5Acreate
- H5Awrite
- H5Aread
- H5Aclose
- H5Aget_name
- H5Aopen_name
- H5Aopen_idx
- H5Aget_space
- H5Aget_type
- H5Aget_num_attrs
- H5Aiterate
- H5Adelete

*Alphabetical Listing*

- H5Aclose
- H5Acreate
- H5Adelete
- H5Aget_name
- H5Aget_num_attrs
- H5Aget_space
- H5Aget_type
- H5Aiterate
- H5Aopen_idx
- H5Aopen_name
- H5Aread
- H5Awrite

*The FORTRAN90 Interfaces:*

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

- h5acreate_f
- h5awrite_f
- h5aread_f
- h5aclose_f
- h5aget_name_f
- h5aget_type_f
- h5aget_num_attrs_f
- h5aget_space_f
- h5aopen_idxF
- h5aopen_name_f
- h5aopen_idx_f
- h5adelete_f

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:

```c
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`

```fortran
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 *name, hid_t type_id, hid_t space_id, hid_t create_plist)
```

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

**Description:**
H5Acreate creates an attribute named `name` and attached to the object specified with `loc_id`. `loc_id` is a group, dataset, or named datatype identifier.

The attribute name specified in `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 H5Acreate.

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, `create_plist`, 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 *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 create_plist` 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:** h5acreate_f
```
SUBROUTINE h5acreate_f(obj_id, name, type_id, space_id, attr_id, &
    hdferr, creation_prp)
    IMPLICIT NONE
    INTEGER(HID_T), INTENT(IN) :: obj_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) :: creation_prp ! Attribute creation property

END SUBROUTINE h5acreate_f
```
Name: H5Adelete

Signature:

    herr_t H5Adelete(hid_t loc_id, const char *name)

Purpose:

Deletes an attribute from a location.

Description:

H5Adelete removes the attribute specified by its name, 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 *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
Name: H5Aget_name

Signature:

\[ \text{size_t } \text{H5Aget_name(hid_t attr_id, size_t buf_size, char *buf) } \]

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.

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 IN: 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, INTENT(IN) :: size            ! Buffer size
  CHARACTER(LEN=*) , INTENT(OUT) :: buf  ! Buffer to hold attribute name
  INTEGER, INTENT(OUT) :: hdferr         ! Error code: name length
                                          ! on success and -1 on failure
END SUBROUTINE h5aget_name_f
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_type  
**Signature:**

```c
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

```fortran
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:

\[
\text{herr_t H5Aiterate(hid_t loc_id, unsigned *idx, H5A_operator_t op, void *op_data)}
\]

Purpose:

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

Description:

H5Aiterate 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:

\[
\text{typedef herr_t (*H5A_operator_t)(hid_t loc_id, const char *attr_name, void *operator_data)}
\]

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 H5Aiterate, 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:

- \text{hid_t loc_id} \quad \text{IN: Identifier of a group, dataset or named datatype.}
- \text{unsigned *idx} \quad \text{IN/OUT: Starting (IN) and ending (OUT) attribute index.}
- \text{H5A_operator_t op} \quad \text{IN: User's function to pass each attribute to}
- \text{void *op_data} \quad \text{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.
**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:**

```c
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 attribute to be attached to.
- `const char *name` **IN:** Attribute name.

**Returns:**

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

**Fortran90 Interface:** h5aopen_name_f

```fortran
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:

\[ \text{herr_t H5Aread(hid_t attr_id, hid_t mem_type_id, void *buf) } \]

Purpose:

Reads an attribute.

Description:

H5Aread reads an attribute, specified with \text{attr_id}. The attribute's memory datatype is specified with \text{mem_type_id}. The entire attribute is read into \text{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 \text{The Data Type Interface (HST)} in the \text{HDF5 User's Guide} for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

Parameters:

\begin{itemize}
  \item \text{hid_t attr_id} \hspace{1cm} \text{IN: Identifier of an attribute to read.}
  \item \text{hid_t mem_type_id} \hspace{1cm} \text{IN: Identifier of the attribute datatype (in memory).}
  \item \text{void *buf} \hspace{1cm} \text{OUT: Buffer for data to be read.}
\end{itemize}

Returns:

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

Fortran90 Interface: h5aread_f

\begin{verbatim}
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
\end{verbatim}

History:

Release Fortran90

1.4.2 The \text{dims} parameter was added in this release.
Name: H5Awrite

Signature:

\[
\text{herr_t H5Awrite(hid_t attr_id, hid_t mem_type_id, const void *buf )}
\]

Purpose:

Writes data to an attribute.

Description:

H5Awrite writes an attribute, specified with \textit{attr_id}. The attribute's memory datatype is specified with \textit{mem_type_id}. The entire attribute is written from \textit{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 \textit{The Data Type Interface (H5T)} in the \textit{HDF5 User's Guide} for a discussion of data conversion, including the range of conversions currently supported by the HDF5 libraries.

Parameters:

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

Returns:

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

Fortran90 Interface: \textsf{h5awrite_f}

\begin{verbatim}
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
\end{verbatim}

History:

Release Fortran90

1.4.2 The \texttt{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
- H5Dopen
- H5Dclose
- H5Dget_space
- H5Dget_space_status
- H5Dget_type
- H5Dget_create_plist
- H5Dget_offset
- H5Dget_storage_size
- H5Dvlen_get_buf_size
- H5Dvlen_reclaim
- H5Dread
- H5Dwrite
- H5Diterate
- H5Dextend
- H5Dfill

Alphabetical Listing

- H5Dclose
- H5Dcreate
- H5Dextend
- H5Dfill
- H5Dget_create_plist
- H5Dget_offset
- H5Dget_space
- H5Dget_space_status
- H5Dget_storage_size
- H5Dget_type
- H5Diterate
- H5Dopen
- H5Dread
- 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.

- h5dcreate_f
- h5dopen_f
- h5dclose_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
- h5dwrite_f
- h5dextend_f
- h5dfill_f
**Name:** H5Dclose  
**Signature:**  
```c
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

```fortran
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 create_plist_id)

Purpose:

Creates a dataset at the specified location.

Description:

H5Dcreate creates a dataset with a name, name, in the file or in the group specified by the identifier loc_id. The dataset has the datatype and dataspace identified by type_id and space_id, respectively. The specified datatype and dataspace are the datatype and dataspace of the dataset as it will exist in the file, which may be different than in application memory. Dataset creation properties are specified by the argument create_plist_id.

Dataset names within a group are unique: H5Dcreate will return an error if a dataset with the name specified in name already exists at the location specified in loc_id.

create_plist_id is a H5P_DATASET_CREATE property list created with H5Pcreate and initialized with the various functions described above.

H5Dcreate returns 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 returns 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.

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_value
- H5Pget_fill_value
- H5Pfill_value_defined
- H5Pset_fill_time
- H5Pget_fill_time
- H5Pset_alloc_time
- 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 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 to all required conditions. This problem is most likely to occur with the use of filters.

For example, H5Dcreate will fail without a meaningful explanation if

-◊ SZIP compression is being used on the dataset and
-◊ 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:**

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>hid_t loc_id</td>
<td>Identifier of the file or group within which to create the dataset.</td>
</tr>
<tr>
<td>const char * name</td>
<td>The name of the dataset to create.</td>
</tr>
<tr>
<td>hid_t type_id</td>
<td>Identifier of the datatype to use when creating the dataset.</td>
</tr>
<tr>
<td>hid_t space_id</td>
<td>Identifier of the dataspace to use when creating the dataset.</td>
</tr>
<tr>
<td>hid_t create_plist_id</td>
<td>Identifier of the set creation property list.</td>
</tr>
</tbody>
</table>

**Returns:**

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

**Fortran90 Interface: h5dcreate_f**

```fortran
SUBROUTINE h5dcreate_f(loc_id, name, type_id, space_id, dset_id, &
                        hdferr, creation_prp)

  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) :: creation_prp
                                     ! Dataset creation property
                                     ! list identifier, default
                                     ! value is H5P_DEFAULT_F (6)

END SUBROUTINE h5dcreate_f
```
Name: H5Dextend

Signature:

herr_t H5Dextend(hid_t dataset_id, const hsize_t *size)

Purpose:
Extends a dataset with unlimited dimension.

Description:
H5Dextend verifies that the dataset is at least of size size. The dimensionality of size is the same as that of the dataspace of the dataset being changed. This function cannot be applied to a dataset with fixed dimensions.

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. (Also see H5Pset_fill_time and H5Pset_alloc_time.)

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
Name: H5Dfill

Signature:

```c
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, and H5Dcreate.

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

```fortran
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
```
Name: H5Dget_create_plist

Signature:

\[
hid_t \text{H5Dget_create_plist}(hid_t \text{dataset_id})
\]

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 for a dataset. The creation property list identifier should be released with the H5Pclose function.

Parameters:

\[
hid_t \text{dataset_id} \quad \text{IN: Identifier of the dataset to query.}
\]

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_prp, 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:**

```c
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

```fortran
SUBROUTINE h5dget_space_status_f(dset_id, flag, hdferr)  
IMPLICIT NONE  
INTEGER(HID_T), INTENT(IN) :: dset_id ! Dataset identifier  
INTEGER, INTENT(IN) :: 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:**  

```c  
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:

```c
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 elements selected in a memory buffer. The callback function is called once for each element selected in the dataspace.

The selection in the dataspace is modified so that any elements already iterated over are removed from the selection if the iteration is interrupted (by the H5D_operator_t function returning non-zero) before the iteration is complete; the iteration may then be re-started by the user where it left off.

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. Also contains the selection to iterate over.
- **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 occurred 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:**  
\[ \text{hid_t H5Dopen(hid_t \ loc\_id, const char *name) } \]  
**Purpose:**  
Opens an existing dataset.  
**Description:**  
H5Dopen 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:**  
- \( \text{hid_t loc\_id} \)  
  IN: Identifier of the file or group within which the dataset to be accessed will be found.  
- \( \text{const char * name} \)  
  IN: The name of the dataset to access.  
**Returns:**  
Returns a dataset identifier if successful; otherwise returns a negative value.  
**Fortran90 Interface:** h5dopen_f  
```fortran  
SUBROUTINE h5dopen_f(loc_id, name, dset_id, hdferr)  
  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  
END SUBROUTINE h5dopen_f  
```
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 datasource identifiers and `H5S_ALL` for the `mem_space_id` and the `file_space_id` parameters is described below:

<table>
<thead>
<tr>
<th><code>mem_space_id</code></th>
<th><code>file_space_id</code></th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid datasource id</td>
<td>valid datasource id</td>
<td><code>mem_space_id</code> specifies the memory dataspace and the selection within it. <code>file_space_id</code> specifies the selection within the file dataset's dataspace.</td>
</tr>
<tr>
<td><code>H5S_ALL</code></td>
<td>valid datasource id</td>
<td>The file dataset's dataspace is used for the memory dataspace and the selection specified with <code>file_space_id</code> specifies the selection within it. The combination of the file dataset's dataspace and the selection from <code>file_space_id</code> is used for memory also.</td>
</tr>
<tr>
<td>valid datasource id</td>
<td><code>H5S_ALL</code></td>
<td><code>mem_space_id</code> specifies the memory dataspace and the selection within it. The selection within the file dataset's dataspace is set to the &quot;all&quot; selection.</td>
</tr>
<tr>
<td><code>H5S_ALL</code></td>
<td><code>H5S_ALL</code></td>
<td>The file dataset's dataspace is used for the memory dataspace and the selection within the memory dataspace is set to the &quot;all&quot; selection. The selection within the file dataset's dataspace is set to the &quot;all&quot; selection.</td>
</tr>
</tbody>
</table>
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:**

```fortran
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 identifier
                   ! Default value is H5S_ALL_F
INTEGER(HID_T), OPTIONAL, INTENT(IN) :: file_space_id
                   ! File dataspace identifier
                   ! 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
```

**History:**

**Release Fortran90**

1.4.2 The `dims` parameter was added in this release.
Name: H5Dvlen_get_buf_size

Signature:

```c
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 counterpart for the C function H5Dvlen_get_buf_size; corresponding
functionality is provided by the FORTRAN function h5dvlen_get_max_len_f.

```fortran
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

<table>
<thead>
<tr>
<th>Version</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.5</td>
<td>Function introduced in this release.</td>
</tr>
</tbody>
</table>

Fortran90

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:

```c
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:

<table>
<thead>
<tr>
<th><code>mem_space_id</code></th>
<th><code>file_space_id</code></th>
<th>Behavior</th>
</tr>
</thead>
<tbody>
<tr>
<td>valid dataspace id</td>
<td>valid dataspace id</td>
<td><code>mem_space_id</code> specifies the memory dataspace and the selection within it. <code>file_space_id</code> specifies the selection within the file dataset's dataspace.</td>
</tr>
<tr>
<td><code>H5S_ALL</code></td>
<td>valid dataspace id</td>
<td>The file dataset's dataspace is used for the memory dataspace and the selection specified with <code>file_space_id</code> specifies the selection within it. The combination of the file dataset's dataspace and the selection from <code>file_space_id</code> is used for memory also.</td>
</tr>
<tr>
<td>valid dataspace id</td>
<td><code>H5S_ALL</code></td>
<td><code>mem_space_id</code> specifies the memory dataspace and the selection within it. The selection within the file dataset's dataspace is set to the &quot;all&quot; selection.</td>
</tr>
<tr>
<td><code>H5S_ALL</code></td>
<td><code>H5S_ALL</code></td>
<td>The file dataset's dataspace is used for the memory dataspace and the selection within the memory dataspace is set to the &quot;all&quot; selection. The selection within the file dataset's dataspace is set to the &quot;all&quot; selection.</td>
</tr>
</tbody>
</table>

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 a 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**

```fortran
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 identifier
   ! Default value is H5S_ALL_F
INTEGER(HID_T), OPTIONAL, INTENT(IN) :: file_space_id ! File dataspace identifier
   ! Default value is H5S_ALL_F
```

47
INTEGER(HID_T), OPTIONAL, INTENT(IN) :: xfer_prp
   ! Transfer property list
   ! identifier; default value
   ! is H5P_DEFAULT_F

END SUBROUTINE h5dwrite_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.

The C Interfaces:

- H5Eclear
- H5Eprint
- H5Epush
- H5Eset_auto
- H5Eget_auto
- H5Ewalk
- H5Ewalk_cb
- H5Eget_major
- H5Eget_minor

Alphabetical Listing

- H5Eclear
- H5Eget_auto
- H5Eget_major
- H5Eget_minor
- H5Eprint
- H5Epush
- H5Eset_auto
- H5Ewalk
- H5Ewalk_cb

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 H5Eprint, 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: H5Eclear
Signature:
\[
herr_t \text{H5Eclear}(\text{void})
\]
Purpose: Clears the error stack for the current thread.
Description: H5Eclear clears the error stack for the current thread.
The stack is also cleared whenever an API function is called, with certain exceptions (for instance, H5Eprint).
H5Eclear can fail if there are problems initializing the library.
Parameters: None
Returns: Returns a non–negative value if successful; otherwise returns a negative value.
Fortran90 Interface: h5eclear_f
SUBROUTINE h5eclear_f(hdferr)
  IMPLICIT NONE
  INTEGER, INTENT(OUT) :: hdferr ! Error code

END SUBROUTINE h5eclear_f
Name: H5Eget_auto

Signature:

\[ \text{herr}_t \text{H5Eget_auto}(\text{H5E_auto}_t \ast \text{func}, \text{void} \ast \ast \text{client_data}) \]

Purpose:

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

Description:

H5Eget_auto returns the current settings for the automatic error stack traversal function, \( \text{func} \), and its data, \( \text{client_data} \). Either (or both) arguments may be null in which case the value is not returned.

Parameters:

- \( \text{H5E_auto}_t \ast \text{func} \) \hspace{1cm} \text{OUT: Current setting for the function to be called upon an error condition.} \\
- \text{void} \ast \ast \text{client_data} \hspace{1cm} \text{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.
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.

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
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.

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
Name: H5Eprint
Signature:
    herr_t H5Eprint(FILE * stream)
Purpose:
    Prints the error stack in a default manner.
Description:
    H5Eprint prints the error stack 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.

    H5Eprint is a convenience function for H5Ewalk 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
    SUBROUTINE h5eprint_f(hdferr, name)
        CHARACTER(LEN=*) , OPTIONAL, INTENT(IN) :: name ! File name
        INTEGER, INTENT(OUT) :: hdferr ! Error code
    END SUBROUTINE h5eprint_f
**Name:** H5Epush

**Signature:**
```c
herr_t H5Epush( 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.

**Description:**
H5Epush 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, file name, 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.
**Name:** H5Eset_auto

**Signature:**

```c
herr_t H5Eset_auto(H5E_auto_t func, void *client_data )
```

**Purpose:**

Turns automatic error printing on or off.

**Description:**

H5Eset_auto 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 `H5Eprint` (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_auto_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

```fortran
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
```
**Name**: H5Ewalk

**Signature**:

```c
herr_t H5Ewalk(H5E_direction_t direction, H5E_walk_t func, void *client_data)
```

**Purpose**:
Walks the error stack for the current thread, calling a specified function.

**Description**:
H5Ewalk 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_walk_t prototype is as follows:

```c
typedef herr_t (*H5E_walk_t)(int n, H5E_error_t *err_desc, void *client_data)
```

where the parameters have the following meanings:

- **int n**: Indexed position of the error in the stack.
- **H5E_error_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.

H5Ewalk can fail if there are problems initializing the library.

**Parameters**:

- **H5E_direction_t direction**  
  IN: Direction in which the error stack is to be walked.
- **H5E_walk_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.
Name: H5Ewalk_cb

Signature:

```c
herr_t H5Ewalk_cb(int n, H5E_error_t *err_desc, void *client_data)
```

Purpose:
Default error stack traversal callback function that prints error messages to the specified output stream.

Description:
H5Ewalk_cb is a default error stack traversal callback function that prints error messages to the specified output stream. It is not meant to be called directly but rather as an argument to the H5Ewalk function. This function is called also by H5Eprint. Application writers are encouraged to use this function as a model for their own error stack walking functions.

n is a counter for how many times this function has been called for this particular traversal of the stack. It always begins at zero for the first error on the stack (either the top or bottom error, or even both, depending on the traversal direction and the size of the stack).

err_desc is an error description. It contains all the information about a particular error.

client_data is the same pointer that was passed as the client_data argument of H5Ewalk. It is expected to be a file pointer (or stderr if NULL).

Parameters:

- `int n`: IN/OUT: Number of times this function has been called for this traversal of the stack.
- `H5E_error_t *err_desc`: OUT: Error description.
- `void *client_data`: IN: A file pointer, or stderr if NULL.

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

Fortran90 Interface:
None.
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_name
- H5Fget_obj_count
- H5Fget_obj_ids
- H5Fget_freespace

Alphabetical Listing

- H5Fclose
- H5Fcreate
- H5FFlush
- H5Fget_create_plist
- H5Fget_access_plist
- H5Fget_name
- H5Fget_obj_count
- H5Fget_obj_ids
- H5Fget_filesize
- H5Fget_freespace
- H5Fis_hdf5
- H5Fmount
- H5Funmount
- H5Fget_vfd_handle

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_create plist_f
- h5fget_access plist_f
- h5fget_name_f
- h5fget_obj_count_f
- h5fget_obj_ids_f
- h5fget_freespace_f
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 explicitly
   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 privilege 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.
Fortran 90 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
**Name:** H5Fcreate  
**Signature:**  
```c
hid_t H5Fcreate(const char *name, unsigned flags, hid_t create_id, hid_t access_id)
```

**Purpose:** Creates HDF5 files.

**Description:**  
H5Fcreate is the primary function for creating HDF5 files.

The `flags` parameter determines whether an existing file will be overwritten. All newly created files are opened for both reading and writing. All flags may be combined with the bit-wise OR operator (`|`) to change the behavior of the H5Fcreate call.

The more complex behaviors of file creation and access are controlled through the file-creation and file-access property lists. The value of H5P_DEFAULT for a property list value indicates that the library should use the default values for the 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.  
- `unsigned 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.  
- An additional flag, `H5F_ACC_DEBUG`, prints debug information. This flag is used only by HDF5 library developers; it is neither tested nor supported for use in applications.
hid_t create_id  IN: File creation property list identifier, used when modifying default file meta-data. Use H5P_DEFAULT for default file creation properties.

hid_t access_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 access_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 property
                           ! 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, new_file_id, 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_pлист

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.


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) :: fcpl_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.


Parameters:

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:**  

```c  
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

```fortran  
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.

- **Fortran90** Fortran subroutine introduced in this release.
**Name:** H5Fget_freespace  
**Signature:**  
  
  \[
  \text{hssize_t H5Fget_freespace(hid_t file_id)}
  \]

**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 a the amount of free space in the file if successful; otherwise returns a negative value.

**Returns:**  
Returns a file creation property list identifier 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  
  
  END SUBROUTINE h5fget_freespace_f

**History:**  
**Release C**  
1.6.1 Function introduced in this release.
Name: H5Fget_name
Signature:

\[
\text{ssize_t \text{H5Fget_name}}(\text{hid_t \text{obj_id}}, \text{char *name, size_t size})
\]

Purpose:

Retrieves name of file to which object belongs.

Description:

H5Fget_name retrieves the name of the file to which the object \text{obj_id} belongs. The object can be a group, dataset, attribute, or named datatype.

Up to \text{size} characters of the filename are returned in \text{name}; additional characters, if any, are not returned to the user application.

If the length of the name, which determines the required value of \text{size}, is unknown, a preliminary H5Fget_name call can be made by setting \text{name} to NULL. The return value of this call will be the size of the filename; that value can then be assigned to \text{size} for a second H5Fget_name call, which will retrieve the actual name.

If an error occurs, the buffer pointed to by \text{name} is unchanged and the function returns a negative value.

Parameters:

hid_t \text{obj_id}

\text{IN: Identifier of the object for which the associated filename is sought. The object can be a group, dataset, attribute, or named datatype.}

char *\text{name}

\text{OUT: Buffer to contain the returned filename.}

size_t \text{size}

\text{IN: Size, in bytes, of the \text{name} buffer.}

Returns:

Returns the length of the filename if successful; otherwise returns a negative value.

Fortran90 Interface: \text{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.

Fortran90

Fortran subroutine introduced in this release.
**Name:** H5Fget_obj_count  

**Signature:**

```c
int 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`)

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 a the number of open objects if successful; otherwise returns a negative value.

**Fortran90 Interface: h5fget_obj_count_f**

```fortran
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_ATTR_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
```
Name: H5Fget_obj_ids

Signature:

    int H5Fget_obj_ids(hid_t file_id, unsigned int types, int 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.

    int 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 C

    1.6.0       Function introduced in this release.
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_for 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:**  
\[ \text{htri_t H5Fis_hdf5}(\text{const char *name}) \]

**Purpose:**  
Determines whether a file is in the HDF5 format.

**Description:**  
H5Fis_hdf5 determines whether a file is in the HDF5 format.

**Parameters:**  
\[ \text{const char *name} \quad \text{IN: File name to check format.} \]

**Returns:**  
When successful, returns a positive value, for TRUE, or 0 (zero), for FALSE. Otherwise 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 location loc_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
Name: H5Fopen

Signature:

hid_t H5Fopen(const char *name, unsigned flags, hid_t access_id)

Purpose:
Opens an existing file.

Description:
H5Fopen opens an existing file and is the primary function for accessing existing HDF5 files.

The parameter access_id is a file access property list identifier or H5P_DEFAULT if the default I/O
access parameters are to be used.

The flags argument determines whether writing to an existing file will be allowed. The file is opened
with read and write permission if flags is set to H5F_ACC_RDWR. All flags may be combined with the
bit-wise OR operator (‘|’) to change the behavior of the file open call. 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 access.
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
  is used only by HDF5 library developers; it is neither tested nor supported
  for use in applications.
hid_t access_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 access_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:** H5Funmount  

**Signature:**  

```c
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`

```fortran
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(IN) :: hdferr ! Error code  
    ! 0 on success and -1 on failure  
END SUBROUTINE h5funmount_f
```
H5G: Group Interface

Group Object API Functions

The Group interface functions create and manipulate groups of objects in an HDF5 file.

The C Interfaces:

- H5Gcreate
- H5Gopen
- H5Gclose
- H5Gset_comment
- H5Gget_comment
- H5Glink
- H5Glink2
- H5Gunlink
- H5Gmove
- H5Gmove2
- H5Giterate
- H5Gget_objinfo
- H5Gget_num_objs
- H5Gget_objname_by_idx
- H5Gget_objtype_by_idx
- H5Gget_linkval

Alphabetical Listing

- H5Gclose
- H5Gcreate
- H5Gget_comment
- H5Gget_linkval
- H5Gget_num_objs
- H5Gget_objinfo
- H5Gget_objname_by_idx
- H5Gget_objtype_by_idx
- H5Giterate
- H5Glink
- H5Glink2
- H5Gunlink
- H5Gmove
- H5Gmove2

The FORTRAN90 Interfaces:

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

- h5gcreate_f
- h5gopen_f
- h5gclose_f
- h5gset_comment_f
- h5gget_comment_f
- h5glink_f
- h5glink2_f
- h5gunlink_f
- h5gmove_f
- h5gmove2_f
- h5gget_obj_info_idx_f
- h5gn_members_f
- h5gget_linkval_f

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.

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

Signature:

hid_t H5Gcreate(hid_t loc_id, const char *name, size_t size_hint)

Purpose:

Creates a new empty group and gives it a name.

Description:

H5Gcreate 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.

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.

Parameters:

- hid_t loc_id: IN: File or group identifier.
- const char *name: IN: Absolute or relative name of the 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: h5gcreate_f

SUBROUTINE h5gcreate_f(loc_id, name, gr_id, hdferr, size_hint)

IMPLICIT NONE

INTEGER(HID_T), INTENT(IN) :: loc_id ! File or group identifier
CHARACTER(LEN=‘), INTENT(IN) :: name ! Name of the group to be created
INTEGER(HID_T), INTENT(OUT) :: gr_id ! Group identifier
INTEGER, INTENT(OUT) :: hdferr ! Error code
! 0 on success and -1 on failure
INTEGER(SIZE_T), OPTIONAL, INTENT(IN) :: size_hint ! Number of bytes to store the names of objects in the group.
! Default value is OBJECT_NAMELEN_DEFAULT_F

END SUBROUTINE h5gcreate_f
**Name:** H5Gget_comment  
**Signature:**  
```c
herr_t H5Gget_comment(hid_t loc_id, const char *name, size_t bufsize, char *comment)
```
**Purpose:**  
Retrieves comment for specified object.
**Description:**  
H5Gget_comment retrieves the comment for the object specified by `loc_id` and `name`. The comment is returned in the buffer `comment`. 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 an object does not have a comment, the empty string is returned.
**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` can be `.’` (dot) if `loc_id` fully specifies the object for which the associated comment is to be retrieved. `name` is ignored if `loc_id` is a dataset or named datatype.
- `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`  
```fortran
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
 END SUBROUTINE h5gget_comment_f
```
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.

Description:  
H5Gget_linkval returns \texttt{size} characters of the name of the object that the symbolic link \texttt{name} points to.

The parameter \texttt{loc_id} is a file or group identifier.

The parameter \texttt{name} must be a symbolic link pointing to the desired object and must be defined relative to \texttt{loc_id}.

If \texttt{size} is smaller than the size of the returned object name, then the name stored in the buffer \texttt{value} will not be null terminated.

This function fails if \texttt{name} is not a symbolic link. The presence of a symbolic link can be tested by passing zero for \texttt{size} and NULL for \texttt{value}.

This function should be used only after H5Gget_objinfo has been called to verify that \texttt{name} is a symbolic link.

Parameters:  
- \texttt{hid_t loc_id} \hspace{1cm} IN: Identifier of the file or group.  
- \texttt{const char *name} \hspace{1cm} IN: Symbolic link to the object whose name is to be returned.  
- \texttt{size_t size} \hspace{1cm} IN: Maximum number of characters of \texttt{value} to be returned.  
- \texttt{char *value} \hspace{1cm} OUT: A buffer to hold the name of the object being sought.

Returns:  
Returns a non-negative value, with the link value in \texttt{value}, if successful. Otherwise returns a negative value.

Fortran90 Interface: \texttt{h5gget_linkval_f}  

SUBROUTINE h5gget_linkval_f(loc_id, name, size, buffer, hdferr)  
IMPLICIT NONE  
INTEGER(HID_T), INTENT(IN) :: loc_id \hspace{1cm} ! File or group identifier  
CHARACTER(LEN=*) , INTENT(IN) :: name \hspace{1cm} ! Name of the symbolic link  
CHARACTER(LEN=size), INTENT(OUT) :: buffer \hspace{1cm} ! Buffer to hold a name of the object  
| CHARACTER(LEN=size), INTENT(OUT) :: hdferr \hspace{1cm} ! Error code  
\hspace{1cm} ! symbolic link points to  
\hspace{1cm} ! 0 on success and -1 on failure  
END SUBROUTINE h5gget_linkval_f
Name: H5Gget_num_objs  
Signature:  
\[ \text{herr}_t \text{H5Gget_num_objs}(\text{hid}_t \text{loc}_id, \text{hsize}_t^* \text{num}_obj) \]  
Purpose:  
Returns number of objects in the group specified by its identifier  
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:  
\[ \begin{align*}  \text{hid}_t \text{loc}_id & \quad \text{IN: Identifier of the group or the file} \\  \text{hsize}_t^* \text{num}_obj & \quad \text{OUT: Number of objects in the group.} \end{align*} \]  
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.
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.

Description:

H5Gget_objinfo returns information about the specified object through the statbuf argument. loc_id (a file or group identifier) and name together determine the object. If the object is a symbolic link and follow_link is zero (0), then the information returned is that for the link itself; otherwise the link is followed and information is returned about 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];
    unsigned long objno[2];
    unsigned nlink;
    int 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 \texttt{mtime} field contains the modification time.

If information is being returned about a symbolic link then \texttt{linklen} will be the length of the link value (the name of the pointed-to object with the null terminator); otherwise \texttt{linklen} will be zero.

The fields in the \texttt{H5O_stat_t} struct contain information about the object header for the object queried:

\begin{itemize}
  \item \texttt{size}  
    The total size of all the object header information in the file (for all chunks).
  \item \texttt{free}  
    The size of unused space in the object header.
  \item \texttt{nmsgs}  
    The number of object header messages.
  \item \texttt{nchunks}  
    The number of chunks the object header is broken up into.
\end{itemize}

Other fields may be added to this structure in the future.

\textbf{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 \texttt{mtime} value of 0 (zero).

\textbf{Parameters:}

\begin{itemize}
  \item \texttt{hid_t loc_id}  
    IN: File or group identifier.
  \item \texttt{const char *name}  
    IN: Name of the object for which status is being sought.
  \item \texttt{hbool_t follow_link}  
    IN: Link flag.
  \item \texttt{H5G_stat_t *statbuf}  
    OUT: Buffer in which to return information about the object.
\end{itemize}

\textbf{Returns:}

Returns a non-negative value if successful, with the fields of \texttt{statbuf} (if non-null) initialized. Otherwise returns a negative value.

\textbf{Fortran90 Interface:}

None.

\textbf{History:}

\begin{itemize}
  \item \textit{Release C}  
    1.6.1 Two new fields were added to the \texttt{H5G_stat_t} struct in this release.
**Name:** H5Gget_objname_by_idx

**Signature:**

```c
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.

**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.
Name: H5Gget_objtype_by_idx

Signature:

\[ \text{int H5Gget_objtype_by_idx(hid_t loc_id, hsize_t idx)} \]

Purpose:

Returns the type of an object specified by an index.

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 The function return type changed from int to the enumerated type H5G_obj_t.
1.6.0 Function introduced 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.

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.

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.

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 counterpart 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.

```fortran
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
```
**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`.

**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:**
```
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
END SUBROUTINE h5glink_f
```
Name: H5Glink2

Signature:

```c
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 )
```

Purpose:

Creates a link of the specified type from `new_name` to `current_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.

Returns:

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

Fortran90 Interface: h5glink2_f

```fortran
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
END SUBROUTINE h5glink2_f
```
**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.

**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
```
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.

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
**Name:** H5Gopen

**Signature:**

hid_t H5Gopen(hid_t loc_id, const char *name)

**Purpose:**

Opens an existing group for modification and returns a group identifier for that group.

**Description:**

H5Gopen opens an existing group with the specified name at the specified location, loc_id.

The location is identified by a file or group identifier

H5Gopen 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: h5gopen_f**

```fortran
SUBROUTINE h5gopen_f(loc_id, name, gr_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: loc_id       ! File or group identifier
  CHARACTER(LEN=*), INTENT(IN) :: name       ! Name of the group to open
  INTEGER(HID_T), INTENT(OUT) :: gr_id       ! Group identifier
  INTEGER, INTENT(OUT) :: hdferr            ! Error code
                                           ! 0 on success and -1 on failure
END SUBROUTINE h5gopen_f
```
Name: \textit{H5Gset_comment} \\
Signature: \\
 \quad \texttt{herr_t H5Gset_comment(hid_t loc_id, const char \*name, const char \*comment)} \\
Purpose: \\
Sets comment for specified object. \\
Description: \\
\textit{H5Gset_comment} sets the comment for the object specified by \texttt{loc_id} and \texttt{name} to \texttt{comment}. Any previously existing comment is overwritten. \\

If \texttt{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, named datatypes, and dataspaces, but not symbolic links. \\

Parameters: \\
\begin{itemize} \\
  \item \texttt{hid_t loc_id} \texttt{IN:} Identifier of the file, group, dataset, or named datatype. \\
  \item \texttt{const char \*name} \texttt{IN:} Name of the object whose comment is to be set or reset. \\
    \texttt{name} can be "." (dot) if \texttt{loc_id} fully specifies the object for which the comment is to be set. \\
    \texttt{name} is ignored if \texttt{loc_id} is a dataset or named datatype. \\
  \item \texttt{const char \*comment} \texttt{IN:} The new comment. \\
\end{itemize} \\

Returns: \\
Returns a non–negative value if successful; otherwise returns a negative value. \\

Fortran90 Interface: \texttt{h5gset_comment_f} \\
\begin{verbatim} \\
SUBROUTINE h5gset_comment_f(loc_id, name, comment, hdferr) \\
  IMPLICIT NONE \\
  INTEGER(HID_T), INTENT(IN) :: loc_id \quad \text{File, group, dataset, or named datatype identifier} \\
  CHARACTER(LEN=*) , INTENT(IN) :: name \quad \text{Name of object} \\
  CHARACTER(LEN=*) , INTENT(IN) :: comment \quad \text{Comment for the object} \\
  INTEGER, INTENT(OUT) :: hdferr \quad \text{Error code} \\
  \text{0 on success and -1 on failure} \\
END SUBROUTINE h5gset_comment_f \\
\end{verbatim}
Name: H5Gunlink

Signature:

```c
herr_t H5Gunlink(hid_t loc_id, const char *name)
```

Purpose:

Removes the link to an object from a group.

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 the 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`

```fortran
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
```
H5I: Identifier Interface

Identifier API Functions

These functions provides tools for working with object identifiers and object names.

The C Interface:

- H5Iget_name
- H5Iget_type
- H5Iget_file_id
- H5Iget_ref
- H5Iinc_ref
- H5Idec_ref

Alphabetical Listing

- H5Idec_ref
- H5Iget_file_id
- H5Iget_name
- H5Iget_ref
- H5Iget_type
- H5Iinc_ref

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

- h5iget_name_f
- h5iget_type_f
- h5iget_file_id_f
- h5iinc_ref_f
- h5idec_ref_f
**Name:** H5Idec_ref  
**Signature:**
```c
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:**
```fortran
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.

- **Fortran90**
  Fortran subroutine introduced in this release.
**Name:** H5Iget_file_id  

**Signature:**  

```c  
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:**  
```fortran  
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.

- **Fortran90**  
  Fortran subroutine introduced in this release.
Name: H5Iget_name

Signature:

\[
\text{ssize_t H5Iget_name(hid_t obj_id, char *name, size_t size)}
\]

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 of the object name. That value can then be assigned to size for a second H5Iget_name call, which will retrieve the actual 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 names, varying according to the path through the HDF5 group hierarchy used to reach that object.

Parameters:

- \text{hid_t obj_id} \hspace{1em} \text{IN: Identifier of the object. This identifier can refer to a group, dataset, or named datatype.}
- \text{char *name} \hspace{1em} \text{OUT: A name associated with the identifier.}
- \text{size_t size} \hspace{1em} \text{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: h5iget_name_f

SUBROUTINE h5iget_name_f(obj_id, buf, buf_size, hdferr)
IMPLICIT NONE
INTEGER(HID_T), INTENT(IN) :: obj_id \hspace{1em} \text{Object identifier}
CHARACTER(LEN=*), INTENT(OUT) :: buf \hspace{1em} \text{Buffer to hold object name}
INTEGER(SIZE_T), INTENT(IN) :: buf_size \hspace{1em} \text{Buffer size}
INTEGER(SIZE_T), INTENT(OUT) :: name_size \hspace{1em} \text{Name size}
INTEGER, INTENT(OUT) :: hdferr \hspace{1em} \text{Error code}
\hspace{1em} - 0 on success, and -1 on failure
END SUBROUTINE h5iget_name_f

History:

Release C

1.6.0 \hspace{1em} 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:**

<table>
<thead>
<tr>
<th>Release</th>
<th>C</th>
<th>Fortran90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6.2</td>
<td>Function introduced in this release.</td>
<td>Fortran subroutine introduced in this release.</td>
</tr>
</tbody>
</table>
**Name:** H5Iget_type  
**Signature:**  

\[ \text{H5I\_type\_t H5Iget\_type(hid\_t obj\_id)} \]

**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

```fortran
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        ! Error code
! 0 on success, and -1 on failure

END SUBROUTINE h5iget_type_f
```
**Name:** H5Iinc_ref

**Signature:**

```c
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`

```fortran
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.

- **Fortran90
  - Fortran subroutine 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.

### The C Interfaces:

#### General Functions

- H5Pcreate
- H5Pget_class
- H5Pcopy
- H5Pclose

#### Generic Properties

- H5Pcreate_class
- H5Pcreate_list
- H5Pregister
- H5Pinsert
- H5Pset
- H5Pexist
- H5Pget_size
- H5Pget_nprops
- H5Pget_class_name
- H5Pget_class_parent
- H5Pisa_class
- H5Pget
- H5Pequal
- H5Piterate
- H5Pcopy_prop
- H5Premove
- H5Punregister
- H5Pclose_list
- H5Pclose_class

#### File Access Properties

- H5Pset_fclose_degree
- H5Pget_fclose_degree
- H5Pset_fapl_core
- H5Pget_fapl_core
- H5Pset_fapl_family
- H5Pget_fapl_family
- H5Pset_family_offset
- H5Pget_family_offset
- H5Pset_fapl_log
- H5Pget_fapl_log

#### File Creation Properties

- H5Pset_userblock
- H5Pget_userblock
- H5Pset_sizes
- H5Pget_sizes
- H5Pset_sym_k
- H5Pget_sym_k
- H5Pset_istore_k
- H5Pget_istore_k

| Function Listing continues on next page. |

|| Indicates functions available only in the parallel HDF5 library. |
### Dataset Creation Properties
- H5Pset_layout
- H5Pget_layout
- H5Pset_chunk
- H5Pget_chunk
- H5Pset_deflate
- H5Pget_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_filter_by_id
- H5Pmodify_filter
- H5Premove_filter
- H5Pset_fletcher32
- H5Pset_shuffle
- H5Pset_szip
- H5Pset_external
- H5Pget_external_count
- H5Pget_external

### Dataset Access, Memory, and Transfer Properties
- H5Pset_buffer
- H5Pget_buffer
- H5Pset_preserve
- H5Pget_preserve
- H5Pset_edc_check
- H5Pget_edc_check
- H5Pset_filter_callback
- H5Pset_hyper_cache *
- H5Pget_hyper_cache *
- 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 ||
- H5Pget_dxpl_mpio ||
- H5Pset_dxpl_multi
- H5Pget_dxpl_multi
- H5Pset_multi_type
- H5Pget_multi_type
- H5Pset_small_data_block_size
- H5Pget_small_data_block_size

---

* Functions labeled with an asterisk (*) are provided only for backwards compatibility with HDF5 Releases 1.4.x. See further notes in the description of each function.
HDF5 Reference Manual

Alphabetical Listing

- H5Pall_filters_avail
- H5Pclose
- H5Pclose_class
- H5Pclose_list
- H5Pcopy
- H5Pcopy_prop
- H5Pcreate
- H5Pcreate_class
- H5Pcreate_list
- H5Pcreate
- H5PClose
- H5PClose_class
- H5PClose_list
- H5PEqual
- H5PExist
- H5PFill_value_defined
- H5PGet
- H5PGet_alignment
- H5PGet_alloc_time
- H5PGet_btree_ratios
- H5PGet_buffer
- H5PGet_cache
- H5PGet_chunk
- H5PGet_class
- H5PGet_class_name
- H5PGet_class_parent
- H5PGet_driver
- H5PGet_dxpl_mpio
- H5PGet_dxpl_multi
- H5PGet_ecHECK
- H5PGet_external
- H5PGet_external_count
- H5PGet_family_offset
- H5PGet_fapl_core
- H5PGet_fapl_family
- H5PGet_fapl_gass
- H5PGet_fapl_gass
- H5PGet_fapl_mpio
- H5PGet_fapl_mpiposix
- H5PGet_fapl_multi
- H5PGet_fapl_split
- H5PGet_fapl_studio
- H5PGet_fapl_stream
- H5PGet_fcloss_degree
- H5PGet_fill_time
- H5PGet_fill_value
- H5PGet_filter
- H5PGet_filter_by_id
- H5PGet_gc_references
- H5PGet_hyper_cache
- H5PGet_hyper_vector_size
- H5PGet_istore_k
- H5PGet_layout
- H5PGet_meta_block_size
- H5PGet_multi_type
- H5PGet_nfilters
- H5PGet_nprops
- H5PGet_preserve
- H5PGet_sieve_buf_size
- H5PGet_version
- H5PGet_userblock
- H5PGet_vlen_mem_manager
- H5Pininsert
- H5PiSaus_class
- H5PIterate
- H5PModify_filter
- H5PRegister
- H5Premove
- H5Premove_filter
- H5Pset
- H5Pset_alignment
- H5Pset_alloc_time
- H5Pset_btree_ratios
- H5Pset_buffer
- H5Pset_cache
- H5Pset_chunk
- H5Pset_deflate
- H5Pset_dxpl_mpio
- H5Pset_dxpl_multi
- H5Pset_ecHECK
- H5Pset_external
- H5Pset_family_offset
- H5Pset_fapl_core
- H5Pset_fapl_family
- H5Pset_fapl_gass
- H5Pset_fapl_log
- H5Pset_fapl_mpio
- H5Pset_fapl_mpiposix
- H5Pset_fapl_multi
- H5Pset_fapl_stream
- H5Pset_fcloss_degree
- H5Pset_fill_time
- H5Pset_fill_value
- H5Pset_filter
- H5Pset_filter_callback
- H5Pset_fletcher32
- H5Pset_gc_references
- H5Pset_hyper_cache
- H5Pset_hyper_vector_size
- H5Pset_istore_k
- H5Pset_layout
- H5Pset_meta_block_size
- H5Pset_multi_type
- H5Pset_preserve
- H5Pset_shuffle
- H5Pset_sieve_buf_size
- H5Pset_sizes
- H5Pset_small_data_block_size
- H5Pset_sym_k
- H5Pset_szip
- H5Pset_userblock
- H5Pset_vlen_mem_manager
- H5Punregister

|| Available only in the parallel HDF5 library.||
The FORTRAN90 Interfaces:
In general, each FORTRAN90 subroutine performs exactly the same task as the corresponding C function.

<table>
<thead>
<tr>
<th>General Property List</th>
<th>File Close Properties</th>
<th>File Access Properties</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Operations</strong></td>
<td><strong>h5pcreate_f</strong></td>
<td><strong>h5pget_driver_f</strong></td>
</tr>
<tr>
<td><strong>h5pget_class_f</strong></td>
<td><strong>h5pget_class_f</strong></td>
<td><strong>h5pset_meta_block_size_f</strong></td>
</tr>
<tr>
<td><strong>h5pcopy_f</strong></td>
<td><strong>h5pcopy_f</strong></td>
<td><strong>h5pset_meta_block_size_f</strong></td>
</tr>
<tr>
<td><strong>h5pclose_f</strong></td>
<td><strong>h5pclose_f</strong></td>
<td><strong>h5pset_sieve_buf_size_f</strong></td>
</tr>
</tbody>
</table>

**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_list_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**

**Dataset Creation Properties**

- **h5pset_layout_f**
- **h5pget_layout_f**
- **h5pset_chunk_f**
- **h5pset_deflate_f**
- **h5pset_fill_value_f**
- **h5pfill_value_defined_f**
- **h5pset_fill_time_f**
- **h5pset_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**
- **h5pmove_filter_f**
- **h5pset_fletcher32_f**
- **h5pset_shuffle_f**
- **h5pset_szip_f**
- **h5pset_external_f**
- **h5pset_external_count_f**
- **h5pset_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_hyper_cache_f**
- **h5pset_hyper_cache_f**
- **h5pset_hyper_vector_size_f**
- **h5pset_hyper_vector_size_f**
- **h5pset_btree_ratios_f**
- **h5pget_btree_ratios_f**
- **h5pset_dxpl_mpio_f**
- **h5pget_dxpl_mpio_f**
- **h5pset_small_data_block_size_f**
- **h5pget_small_data_block_size_f**

Available only in the parallel HDF5 library.
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: H5Pclose_list

Signature:

herr_t H5Pclose_list(hid_t plist)

Purpose:
Closes a property list.

Description:

H5Pclose_list closes a property list.

If a close callback exists for the property list class, it is called before the property list is destroyed. If close callbacks exist for any individual properties in the property list, they are called after the class close callback.

Parameters:

hid_t plist IN: Property list to close

Returns:

Success: a non–negative value
Failure: a negative value

Fortran90 Interface: h5pclose_list_f

SUBROUTINE h5pclose_list_f(plist, hdferr)
IMPLICIT NONE
INTEGER(HID_T), INTENT(IN) :: plist ! Property list identifier to close
INTEGER, INTENT(OUT) :: hdferr ! Error code
! 0 on success and -1 on failure
END SUBROUTINE h5pclose_list_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`

```fortran
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 to copy
INTEGER, INTENT(OUT) :: hdferr             ! Error code
                                      ! 0 on success and -1 on failure
END SUBROUTINE h5pcopy_f
```
Name: H5Pcopy_prop

Signature:

\[ \text{herr_t H5Pcopy_prop( hid_t dst_id, hid_t src_id, const char *name )} \]

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 H5Pininsert (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:
- \( \text{hid_t dst_id} \) IN: Identifier of the destination property list or class
- \( \text{hid_t src_id} \) IN: Identifier of the source property list or class
- \( \text{const char *name} \) IN: Name of the property to copy

Returns:
Success: a non-negative value
Failure: a negative value

Fortran90 Interface: h5pcopy_prop_f

```fortran
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:**  
```c
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:

**H5P_FILE_CREATE**  
Properties for file creation. See Files in the *HDF User's Guide* for details about the file creation properties.

**H5P_FILE_ACCESS**  
Properties for file access. See Files in the *HDF User's Guide* for details about the file creation properties.

**H5P_DATASET_CREATE**  
Properties for dataset creation. See Datasets in the *HDF User's Guide* for details about dataset creation properties.

**H5P_DATASET_XFER**  
Properties for raw data transfer. See Datasets in the *HDF User's Guide* for details about raw data transfer properties.

**H5P_MOUNT**  
Properties for file mounting. With this parameter, H5Pcreate creates and returns a new mount property list initialized with default values.

This property list must eventually be closed with H5Pclose; otherwise, errors are likely to occur.

**Parameters:**  
```c
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  
```fortran
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
```
**Name:** H5Pcreate_class

**Signature:**

```c
hid_t H5Pcreate_class( hid_t class, const char *name, H5P_cls_create_func_t create, H5P_cls_copy_func_t copy, H5P_cls_close_func_t close )
```

**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 or may be derived from the default "empty" class. New classes with inherited properties from existing classes may not remove those existing properties, only add or remove their own class properties. 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:

```c
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/OUT: User pointer to any class creation information needed

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:

```c
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/OUT: User pointer to any class copy information needed

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:

```c
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/OUT: User pointer to any class close information needed

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 class` IN: Property list class to inherit from.
- `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
- `H5P_cls_copy_func_t copy` IN: Callback routine called when a property list is copied
- `H5P_cls_close_func_t close` IN: Callback routine called when a property list is being closed

**Returns:**

Success: a valid property list class identifier

Failure: 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: H5Pcreate_list

Signature:

hid_t H5Pcreate_list(hid_t class)

Purpose:

Creates a new property list class of a given class.

Description:

H5Pcreate_list creates a new property list of a given class. If a create callback exists for the property list class, it is called before the property list is passed back to the user. If create callbacks exist for any individual properties in the property list, they are called before the class create callback.

Parameter:

hid_t class; IN: Class of property list to create.

Returns:

Success: a valid property list identifier
Failure: a negative value

Fortran90 Interface:

None.
**Name:** H5Premove_filter  
**Signature:**  
\[ \text{herr_t H5Premove_filter(hid_t plist, H5Z_filter_t filter)} \]  
**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
- H5Z_FILTER_DEFLATE
- H5Z_FILTER_SHUFFLE
- H5Z_FILTER_FLETCHER32
- H5Z_FILTER_SZIP


INTEGER, INTENT(OUT) :: hdferr
!   Error code
!   0 on success, -1 on failure
END SUBROUTINE h5premove_filter_f

History:

<table>
<thead>
<tr>
<th>Release</th>
<th>C</th>
<th>Fortran90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6.3</td>
<td>Function introduced in this release.</td>
<td>Fortran subroutine introduced in this release.</td>
</tr>
</tbody>
</table>
**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, INTENT(INOUT) ::= 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:**

```c
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 the property

**Returns:**

- Success: a non–negative value
- Failure: a negative value

**Fortran90 Interface: h5pget_f**

```fortran
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:

```c
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 `fill_time` as one of the following values:

- **H5D_ALLOC_TIME_DEFAULT**: Uses the default allocation time, based on the dataset storage method.
  See the `fill_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

```fortran
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
INTEGER, INTENT(OUT) :: hdferr ! Error code
END SUBROUTINE h5pget_alloc_time_f
```

History:

**Release C**

1.6.0 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:

\[ \text{hsize_t} \text{H5Pget_buffer(hid_t plist, void **tconv, void **bkg)} \]

Purpose:

Reads buffer settings.

Description:

H5Pget_buffer reads values previously set with H5Pset_buffer.

Parameters:

- \text{hid_t} \text{plist} \hspace{1cm} \text{IN: Identifier for the dataset transfer property list.}
- \text{void **tconv} \hspace{1cm} \text{OUT: Address of the pointer to application−allocated type conversion buffer.}
- \text{void **bkg} \hspace{1cm} \text{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
END SUBROUTINE h5pget_buffer_f

History:

- \text{Release C}
  - 1.6.0 The return type changed from \text{hsize_t} to \text{size_t}.
  - 1.4.0 The return type changed to \text{hsize_t}. 
**Name:** H5Pget_cache

**Signature:**

```c
herr_t H5Pget_cache(hid_t plist_id, int *mdc_nelmts, int *rdcc_nelmts, size_t *rdcc_nbytes, double *rdcc_w0)
```

**Purpose:**
Queries the meta data cache and raw data chunk cache parameters.

**Description:**
H5Pget_cache retrieves the maximum possible number of elements in the meta data cache and 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.

**Parameters:**
- `hid_t plist_id`
  IN: Identifier of the file access property list.
- `int *mdc_nelmts`
  IN/OUT: Number of elements (objects) in the meta data cache.
- `int *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

```fortran
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 raw data chunk 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**

1.6.0 The rdcc_nbytes parameter changed from type `int` to `size_t`. 

---

**HDF5 Reference Manual H5Pget_cache 133**
**Name:** H5Pget_chunk

**Signature:**

```c
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 successful; otherwise returns a negative value.

**Fortran90 Interface:** h5pget_chunk_f

```fortran
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
```
Name: H5Pget_class

Signature:

\[ \text{H5P_class_t H5Pget_class(hid_t \text{plist})} \]

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 \text{plist} parameter. Valid property list classes are defined in the description of H5Pcreate.

Parameters:

\text{hid_t \text{plist}} \quad \text{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:**
```c
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

```fortran
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:**

\[ \text{hid_t H5Pget_class_parent(hid_t pcid)} \]

**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:**

- \( \text{hid_t pcid} \) 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

```fortran
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_driver

Signature:

\[ \text{hid}_t \text{H5Pget}\_driver(\text{hid}_t \text{plist\_id}) \]

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 \text{plist\_id}.

Valid driver identifiers with the standard HDF5 library distribution include the following:

\begin{verbatim}
H5FD_CORE
H5FD_FAMILY
H5FD_GASS
H5FD_LOG
H5FD_MPI0
H5FD_MULTI
H5FD_SEC2
H5FD_STDIO
H5FD_STREAM
\end{verbatim}

If a user defines and registers custom drivers or if additional drivers are defined in an HDF5 distribution, this list will be longer.

The returned driver identifier is only valid as long as the file driver remains registered.

Parameters:

\text{hid}_t \text{plist\_id} \quad \text{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

\begin{verbatim}
SUBROUTINE h5pget_driver\_f(prp\_id, driver, hdferr)
IMPLICIT NONE
INTEGER(HID_T), INTENT(IN) :: prp\_id \quad \text{Property list identifier}
INTEGER, INTENT(OUT) :: driver \quad \text{Low−level file driver identifier}
INTEGER, INTENT(OUT) :: hdferr \quad \text{Error code}
END SUBROUTINE h5pget_driver\_f
\end{verbatim}

History:

Release C

1.4.0 Function introduced in this release.
**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:

\[ H5Z\_EDC\_t \, H5Pget\_edc\_check(hid\_t\, \text{plist}) \]

Purpose:
Determines whether error-detection is enabled for dataset reads.

Description:
H5Pget_edc_check queries the dataset transfer property list \( \text{plist} \) to determine whether error detection is enabled for data read operations.

Parameters:

\[ \text{hid\_t\, \text{plist}} \]
IN: Dataset transfer property list identifier.

Returns:
Returns \( H5Z\_\text{DISABLE\_EDC} \) or \( H5Z\_\text{ENABLE\_EDC} \) 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.
Name: H5Pget_external

Signature:

```c
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

```fortran
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, 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`.

142
Name: H5Pget_external_count

Signature:

\[
\text{int H5Pget_external_count(hid_t plist)}
\]

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:

\[
hid_t plist \quad \text{IN: Identifier of a dataset creation property list.}
\]

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:**

\[ \text{herr_t H5Pget_family_offset (hid_t fapl_id, hsize_t *offset)} \]

**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` IN: 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_family

Signature:

```c
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

```fortran
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.
Name: H5Pget_fapl_gass
Signature:

    herr_t H5Pget_fapl_gass( hid_t fapl_id, GASS_Info *info )

Purpose:
Retrieves GASS information.

Description:
If the file access property list fapl_id is set for use of the H5FD_GASS driver, H5Pget_fapl_gass returns the GASS_Info object through the info pointer.

The GASS_Info information is copied, so it is valid only until the file access property list is modified or closed.

Note:
H5Pget_fapl_gass is an experimental function. It is designed for use only when accessing files via the GASS facility of the Globus environment. For further information, see http://www.globus.org/.

Parameters:

hid_t fapl_id, IN: File access property list identifier.
GASS_Info *info OUT: Pointer to the GASS information structure.

Returns:
Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface:
None.
Name: H5Pget_fapl_mpio

Signature:

\[
\text{herr_t H5Pget_fapl_mpio( hid_t fapl_id, MPI_Comm *comm, MPI_Info *info )}
\]

Purpose:

Returns MPI communicator information.

Description:

If the file access property list is set to the H5FD_MPIO driver, H5Pget_fapl_mpio returns the MPI communicator and information through the comm and info pointers, if those values are non-null.

Neither comm nor info is copied, so they are valid only until the file access property list is either modified or closed.

Parameters:

- \(hid_t fapl_id\)  
  IN: File access property list identifier.
- \(MPI\_Comm *\text{comm}\)  
  OUT: MPI-2 communicator.
- \(MPI\_Info *\text{info}\)  
  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
  INTEGER, INTENT(IN) :: info ! 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 C

1.4.5 This function's handling of the MPI Communicator and Info objects changed at this release. A copy of each of these objects is now stored in the property list instead of pointers to each object.

1.4.0 Function introduced in this release.
Name: H5Pget_fapl_mpiposix

Signature:

    herr_t H5Pget_fapl_mpiposix(hid_t fapl_id, MPI_Comm *comm)

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.

Parameters:

    hid_t fapl_id        IN: File access property list identifier.
    MPI_Comm *comm       OUT: MPI-2 communicator.

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      

    1.6.0      Function introduced in this release.
    1.6.0      A use_gpfs parameter of type hbool_t has been added.
**Name:** H5Pget_fapl_multi

**Signature:**

```c
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:

- `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.

**Fortran Interface:** h5pget_fapl_multi_f

```fortran
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

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_fapl_srb

Signature:

```c
herr_t H5Pget_fapl_srb( hid_t fapl_id, SRB_Info *info )
```

Purpose:
Retrieves SRB information.

Description:
If the file access property list `fapl_id` is set for use of the `H5FD_SRB` driver, `H5Pget_fapl_srb` returns the `SRB_Info` object through the `info` pointer.

The `SRB_Info` information is copied, so it is valid only until the file access property list is modified or closed.

Note:
`H5Pset_fapl_gass` is an experimental function. It is designed for use only when accessing files via the Storage Resource Broker (SRB). For further information, see http://www.npaci.edu/Research/DI/srb/.

Parameters:
- `hid_t fapl_id` IN: File access property list identifier.
- `SRB_Info *info` OUT: Pointer to the SRB information structure.

Returns:
Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface:
None.
Name: H5Pget_fapl_stream

Signature:

```
herr_t H5Pget_fapl_stream( hid_t fapl_id, H5FD_stream_fapl_t *fapl )
```

Purpose:

Returns the streaming I/O driver settings.

Description:

H5Pget_fapl_stream returns the file access properties set for the use of the streaming I/O driver.

H5Pset_fapl_stream and H5Pget_fapl_stream are not intended for use in a parallel environment.

Parameters:

- `hid_t fapl_id` IN: File access property list identifier.
- `H5FD_stream_fapl_t *fapl` OUT: The streaming I/O file access property list.

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_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 \texttt{fc\_degree} in the file access property list \texttt{fapl\_id}.

The value of \texttt{fc\_degree} determines how aggressively H5Fclose deals with objects within a file that remain open when H5Fclose is called to close that file. \texttt{fc\_degree} can have any one of four valid values as described above in H5Pset_fclose_degree.

**Parameters:**

- \texttt{hid_t fapl\_id}  
  IN: File access property list identifier.
- \texttt{H5F\_close\_degree\_t *fc\_degree}  
  OUT: Pointer to a location to which to return the file close degree property, the value of \texttt{fc\_degree}.

**Returns:**  
Returns a non-negative value if successful. Otherwise returns a negative value.

**Fortran90 Interface:** h5pget_fclose_degree_f

```fortran
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:

\[ \text{herr_t H5Pget_fill_time(hid_t plist_id, H5D_fill_time_t *fill_time)} \]

Purpose:
Retrieves the time when fill value are written to a dataset.

Description:
H5Pget_fill_time examines the dataset creation property list \( \text{plist_id} \) to determine when fill values are to be written to a dataset.

Valid values returned in \( \text{fill_time} \) are as follows:

- \text{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)

- \text{H5D_FILL_TIME_ALLOC}:
  Fill values are written to the dataset when storage space is allocated.

- \text{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:

- \text{hid_t plist_id} IN: Dataset creation property list identifier.
- \text{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  

**Signature:**

```c
H5Z_filter_t H5Pget_filter(hid_t plist, int filter_number, 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.

**Description:**

H5Pget_filter 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.

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.

`filter_number` 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.

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.

The structure of the `flags` argument is discussed in H5Pset_filter.

**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 filter_number`  
  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

Otherwise returns a negative value.

**Fortran90 Interface: h5pget_filter_f**

```fortran
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.
Name: H5Pget_filter_by_id

Signature:

```c
herr_t H5Pget_filter_by_id(hid_t plist_id, H5Z_filter_t filter, unsigned int *flags,
                         size_t *cd_nelmts, unsigned int cd_values[], size_t namelen, char name[])
```

Purpose:

Returns information about the specified filter.

Description:

H5Pget_filter_by_id returns information about the filter specified in `filter`, a filter identifier.

`plist_id` must identify a dataset creation property list and `filter` will be in a permanent filter pipeline.

The `filter` 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` is not set for the property list, an error will be returned and H5Pget_filter_by_id will fail.

Parameters:

- `hid_t plist_id` IN: Property list identifier.
- `H5Z_filter_t filter` 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[]`.
- `const 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

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**

1.6.0 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_cache

Signature:

\[
\text{herr_t } \text{H5Pget_hyper_cache}(\text{hid_t } \text{plist}, \text{unsigned } * \text{cache}, \text{unsigned } * \text{limit})
\]

Purpose:

[NOTE: This function is deprecated in HDF5 Release 1.6 and will eventually be removed from the HDF5 distribution. It is provided in this release only to enable backward compatibility with HDF5 Releases 1.4.x and is enabled only if the HDF5 library is configured with the flag H5_WANT_H5_V1_4_COMPAT; the function is not enabled in the binaries distributed by NCSA.]

Returns information regarding the caching of hyperslab blocks during I/O.

Description:

Given a dataset transfer property list, H5Pget_hyper_cache returns instructions regarding the caching of hyperslab blocks during I/O. These parameters are set with the H5Pset_hyper_cache function.

Parameters:

hid_t plist

IN: Dataset transfer property list identifier.

unsigned *cache

OUT: A flag indicating whether caching is set to on (1) or off (0).

unsigned *limit

OUT: Maximum size of the hyperslab block to cache. 0 (zero) indicates no limit.

Returns:

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

Fortran90 Interface: h5pget_hyper_cache_f

SUBROUTINE h5pget_hyper_cache_f(prp_id, cache, limit, hdferr)
IMPLICIT NONE
INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
INTEGER, INTENT(OUT) :: cache !
INTEGER, INTENT(OUT) :: limit ! Maximum size of the hyperslab block to cache
! 0 (zero) indicates no limit
INTEGER, INTENT(OUT) :: hdferr ! Error code
! 0 on success and -1 on failure
END SUBROUTINE h5pget_hyper_cache_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:**  
\[HSD\textunderscore layout\_t H5Pget\_layout(hid\_t\ plist)\]  
**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:**  
```
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_meta_block_size

Signature:

\[ \text{herr_t H5Pget_meta_block_size}(\text{hid_t fapl_id, hsize_t *size}) \]

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:

- \text{hid_t fapl_id} \hspace{1em} \text{IN: File access property list identifier.}
- \text{hsize_t *size} \hspace{1em} \text{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 H5Pset_multi_type (hid_t fapl_id, H5FD_mem_t *type)
```

Purpose:
Retrieves data type property for MULTI driver.

Description:
H5Pget_multi_type retrieves the data type 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_nprops  
**Signature:**  

```c
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  
```fortran
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.

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_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:**

```c
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

```fortran
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 *sizeof_addr` OUT: Pointer to location to return offset size in bytes.
- `size_t *sizeof_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:**  
\[ \text{herr_t H5Pget_small_data_block_size(hid_t fapl_id, hsize_t *size) } \]

**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:**
- \( \text{hid_t fapl_id} \) IN: File access property list identifier.
- \( \text{hsize_t *size} \) 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

```fortran
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
END SUBROUTINE h5pget_small_data_block_size_f
```

**History:**

**Release C**

1.4.4 Function introduced in this release.
**Name:** H5Pget_sym_k

**Signature:**

```c
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 *lk` 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`

```fortran
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_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:

```c
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

```fortran
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  

**Signature:**

```
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 )
```

**Purpose:** Registers a temporary property with a property list.

**Description:**

H5Pinsert 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 );
```

where the parameters to the 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); where the parameters to the 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); where the parameters to the 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 ID 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:** `h5insert_f`

```fortran
SUBROUTINE h5insert_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 h5insert_f
```
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 a property list is a member of the specified class.

Parameters:
- `hid_t plist` IN: Identifier of the property list
- `hid_t pclass` IN: Identifier of the property class

Returns:
- Success: TRUE (positive) if equal; FALSE (zero) if unequal
- Failure: a negative value

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:**
```c
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. *(NOTE: `iter_func` was changed to `H5P_iterate_t` in the preceding sentence. Is this correct?)*

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:**
```c
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:**  
```c
herr_t H5Pmodify_filter(hid_t plist, H5Z_filter_t filter, 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 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`, `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` 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  
```fortran
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
t 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

### Signature:

```c
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 )
```

### Purpose:

Registers a permanent property with a property list class.

### Description:

H5Pregister 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:

```c
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 H5Pregister.

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:

```c
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:

```c
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:

```c
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:

```c
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:

```c
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:
The identifier of the property list being closed

IN: The name of the property in the list

const char *name

IN: The size of the property in bytes

size_t size

IN: The value for the property being closed

void *value

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: h5register_f

SUBROUTINE h5register_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 h5register_f
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 H5Pinsert) 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:** H5Pset

**Signature:**

```c
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

```fortran
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:

\[
\text{herr_t H5Pset_alignment(hid_t \text{plist}, hsize_t \text{threshold}, hsize_t \text{alignment})}
\]

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:
- \text{id_t \text{plist}}  \text{IN: Identifier for a file access property list.}
- \text{hsize_t \text{threshold}}  \text{IN: Threshold value. Note that setting the threshold value to 0 (zero) has the effect of a special case, forcing everything to be aligned.}
- \text{hsize_t \text{alignment}}  \text{IN: Alignment value.}

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

Fortran90 Interface: H5pset_alignment_f

```fortran
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:

\[ \text{herr_t H5Pset_alloc_time(hid_t plist_id, H5D_alloc_time_t alloc_time) } \]

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 fill_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

```fortran
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_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:**
```
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:**
```c
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
```fortran
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
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`. 
Name: H5Pset_cache

Signature:

    herr_t H5Pset_cache(hid_t plist_id, int mdc_nelmts, int rdcc_nelmts, size_t rdcc_nbytes, double rdcc_w0)

Purpose:
Sets the meta data cache and raw data chunk cache parameters.

Description:
H5Pset_cache sets the number of elements (objects) in the meta data cache and 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 meta data cache and the raw data chunk cache are mdc_nelmts and rdcc_nelmts, respectively. The total size of the raw data chunk cache and the preemption policy are rdcc_nbytes and rdcc_w0.

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.

Parameters:

hid_t plist_id 
IN: Identifier of the file access property list.

int mdc_nelmts 
IN: Number of elements (objects) in the meta data cache.

int 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, INTENT(IN) :: rdcc_nelmts ! Number of elements (objects)
! in the raw data chunk 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 C

1.6.1
rdcc_nbytes parameter type changed to INTEGER(SIZE_T).

1.6.0 The rdcc_nbytes parameter has changed from type int to size_t.
**Name:** H5Pset_chunk  
**Signature:**  
```c  
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.)

**Parameters:**
- `hid_t plist` IN: Identifier for property list to query.
- `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:**  
```fortran  
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  
```
**Name:** H5Pset_deflate

**Signature:**
```
herr_t H5Pset_deflate(hid_t plist, int 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.
- `int 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_dxpl_mpio  
**Signature:**

```c
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:**

```fortran
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_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:

\texttt{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 \texttt{plist} to enable or disable error detection when reading data.

Whether error detection is enabled or disabled is specified in the \texttt{check} parameter. Valid values are as follows:

- \texttt{H5Z_ENABLE_EDC} (default)
- \texttt{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:

- \texttt{hid_t plist} IN: Dataset transfer property list identifier.
- \texttt{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: \texttt{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.
**Name:** H5Pset_external  
**Signature:**

```c
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

```fortran
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:**

```c
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.

If `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.

**Note:**

There is currently no means for reading a file from disk then using the H5FD_CORE driver to manipulate the file.

**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

```fortran
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:**

<table>
<thead>
<tr>
<th>Release</th>
<th>C</th>
<th>Fortran90</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.6.0</td>
<td></td>
<td>The <code>backing_store</code> parameter has changed from <code>INTEGER</code> to <code>LOGICAL</code> to better match the C API.</td>
</tr>
<tr>
<td>1.4.0</td>
<td>Function introduced in this release.</td>
<td></td>
</tr>
</tbody>
</table>
**Name:** H5Pset_fapl_family  
**Signature:**  
```c
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. Because this size is not saved in the file, it is used both for creating a new file, for re-opening and for extending an existing file.

When re-opening an existing family file, if there is only one member file, the library allows this `memb_size` to be bigger than or equal to the size of existing member file; if there are more than one member file, the library sets the `memb_size` to be equal to the size of first existing member file internally. In either case, no `memb_size` smaller than the size of existing member file is allowed. If this happens, the library will adjust the `memb_size` to the first existing member file size internally instead of returning error.

For example, if the total file size is 1MB and the only existing member file size is 1MB, `memb_size` can be bigger than or equal to 1MB. If the first member file size is 0.6MB and the second one is 0.4MB, the library will set `memb_size` to 0.6MB internally no matter what value the user passes in.

`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  
```fortran
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) :: imemb_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_gass**

**Signature:**

```c
herr_t H5Pset_fapl_gass(hid_t fapl_id, GASS_Info info)
```

**Purpose:**

Stores user-supplied GASS information.

**Description:**

`H5Pset_fapl_gass` stores user-supplied GASS information, the `GASS_Info` struct data as passed in `info`, to the file access property list `fapl_id`. `fapl_id` can then be used to create and/or open the file.

The `GASS_Info` object, `info`, is used for file open operations when using GASS in the Globus environment.

Any modification to `info` after this function call returns may have undetermined effect to the access property list. Users must call `H5Pset_fapl_gass` again to setup the property list.

**Note:**

`H5Pset_fapl_gass` is an experimental function. It is designed for use only when accessing files via the GASS facility of the Globus environment. For further information, see [http://www.globus.org/](http://www.globus.org/).

**Parameters:**

- `hid_t fapl_id`, IN: File access property list identifier.
- `GASS_Info info` IN: Pointer to the GASS information structure.

**Returns:**

Returns a non-negative value if successful. Otherwise returns a negative value.

**Fortran90 Interface:**

None.
**Name:** H5Pset_fapl_log  
**Signature:**

```c
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).

<table>
<thead>
<tr>
<th>Flag</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H5FD_LOG_LOC_READ</td>
<td>Track the location and length of every read, write, or seek operation.</td>
</tr>
<tr>
<td>H5FD_LOG_LOC_WRITE</td>
<td>Track all I/O locations and lengths. The logical equivalent of the following:</td>
</tr>
<tr>
<td></td>
<td>(H5FD_LOG_LOC_READ</td>
</tr>
<tr>
<td>H5FD_LOG_LOC_SEEK</td>
<td></td>
</tr>
<tr>
<td>H5FD_LOG_LOC_IO</td>
<td>Track all I/O locations and lengths. The logical equivalent of the following:</td>
</tr>
<tr>
<td></td>
<td>(H5FD_LOG_LOC_READ</td>
</tr>
<tr>
<td>H5FD_LOG_FILE_READ</td>
<td>Track the number of times each byte is read or written.</td>
</tr>
<tr>
<td>H5FD_LOG_FILE_WRITE</td>
<td>Track the number of times each byte is read and written. The logical equivalent of the following:</td>
</tr>
<tr>
<td></td>
<td>(H5FD_LOG_FILE_READ</td>
</tr>
<tr>
<td>H5FD_LOG_FILE_IO</td>
<td></td>
</tr>
<tr>
<td>H5FD_LOG_FLAVOR</td>
<td>Track the type, or flavor, of information stored at each byte.</td>
</tr>
<tr>
<td>H5FD_LOG_NUM_READ</td>
<td>Track the total number of read, write, or seek operations that occur.</td>
</tr>
<tr>
<td>H5FD_LOG_NUM_WRITE</td>
<td>Track the total number of all types of I/O operations. The logical equivalent of the following:</td>
</tr>
<tr>
<td></td>
<td>(H5FD_LOG_NUM_READ</td>
</tr>
<tr>
<td>H5FD_LOG_NUM_SEEK</td>
<td></td>
</tr>
<tr>
<td>H5FD_LOG_NUM_IO</td>
<td></td>
</tr>
</tbody>
</table>
H5FD_LOG_TIME_OPEN  
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_READ

H5FD_LOG_TIME_WRITE

H5FD_LOG_TIME_SEEK

H5FD_LOG_TIME_CLOSE

H5FD_LOG_TIME_IO

Track the time spent in each of the above operations. The logical equivalent of the following:  
\[(H5FD_{\text{LOG\_TIME\_OPEN}} \mid H5FD_{\text{LOG\_TIME\_READ}} \mid H5FD_{\text{LOG\_TIME\_WRITE}} \mid H5FD_{\text{LOG\_TIME\_SEEK}} \mid H5FD_{\text{LOG\_TIME\_CLOSE}})\]

H5FD_LOG_ALLOC

Track the allocation of space in the file.

H5FD_LOG_ALL

Track everything. The logical equivalent of the following:  
\[(H5FD_{\text{LOG\_ALLOC}} \mid H5FD_{\text{LOG\_TIME\_IO}} \mid H5FD_{\text{LOG\_NUM\_IO}} \mid H5FD_{\text{LOG\_FLAVOR}} \mid H5FD_{\text{LOG\_FILE\_IO}} \mid H5FD_{\text{LOG\_LOC\_IO}})\]

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.
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 the 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 does not create a duplicated communicator. Modifications to comm after this function call returns may have an undetermined effect on the access property list. Users should not modify the communicator while it is defined in a 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 does not create a duplicated info object. Any modification to the info object after this function call returns may have an undetermined effect on the access property list. Users should not modify the info while it is defined in a property list.

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 C

1.4.5  This function's handling of the MPI Communicator and Info objects changed at this release. A copy of each of these objects is now stored in the property list instead of pointers to each object.

1.4.0  Function introduced in this release.
**Name:** H5Pset_fapl_mpiposix

**Signature:**

```
    herr_t H5Pset_fapl_mpiposix(hid_t fapl_id, MPI_Comm comm)
```

**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 access property list. Users should not modify the communicator while it is defined in a property list.

**Parameters:**

- `hid_t fapl_id` IN: File access property list identifier.
- `MPI_Comm comm` IN: MPI-2 communicator.

**Returns:**

Returns a non-negative value if successful. Otherwise returns a negative value.

**Fortran90 Interface:**

```
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 C**

- **1.6.0** A `use_gpfs` parameter of type `hbool_t` has been added.
- **1.6.0** Function introduced in this release.

**Fortran90**

Fortran subroutine 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`

```fortran
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

H5Pset_fapl_multi HDF5 Reference Manual
210
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:

```c
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

```fortran
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:**  
```c
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:**
```c
/* 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",H5P_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, &
   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
   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_srb

Signature:

    herr_t H5Pset_fapl_srb( hid_t fapl_id, SRB_Info info )

Purpose:

Saves SRB connection handler and sets SRB settings.

Description:

H5Pset_fapl_srb stores the SRB client-to-server connection handler SRB_CONN after the connection is established and other user-supplied SRB information.

The user-supplied SRB information is contained in the SRB_Info struct pointed to by info and is stored in the file access property list fapl_id. This information can then be used to create or open a file.

Note:

H5Pset_fapl_gass is an experimental function. It is designed for use only when accessing files via the Storage Resource Broker (SRB). For further information, see http://www.npaci.edu/Research/DI/srb/.

Parameters:

hid_t fapl_id       IN: File access property list identifier.
SRB_Info info      IN: Pointer to the SRB information structure.

Returns:

Returns a non-negative value if successful. Otherwise returns a negative value.

Fortran90 Interface:

None.
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_stream  
**Signature:**
```c
herr_t H5Pset_fapl_stream( hid_t fapl_id, H5FD_stream_fapl_t *fapl )
```

**Purpose:**
Sets up the use of the streaming I/O driver.

**Description:**
H5Pset_fapl_stream sets up the use of the streaming I/O driver.

fapl_id is the identifier for the file access property list currently in use.

fapl is the file access property list.

The H5FD_stream_fapl_t struct contains the following elements:

- `size_t increment`
- `H5FD_STREAM_SOCKET_TYPE socket`
- `hbool_t do_socket_io`
- `unsigned int backlog`
- `H5FD_stream_broadcast_t broadcast_fn`
- `void * broadcast_arg`

- `increment` specifies how much memory to allocate each time additional memory is required.
- `socket` is an external socket descriptor; if a valid socket argument is provided, that socket will be used.
- `do_socket_io` is a boolean value specifying whether to perform I/O on `socket`.
- `backlog` is the argument for the `listen` call.
- `broadcast_fn` is the broadcast callback function.
- `broadcast_arg` is the user argument to the broadcast callback function.

H5Pset_fapl_stream and H5Pget_fapl_stream are not intended for use in a parallel environment.

**Parameters:**
- `hid_t fapl_id` IN: File access property list identifier.
- `H5FD_stream_fapl_t *fapl` IN: The streaming I/O file access property list.

**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_fclose_degree

Signature:

```c
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:

<table>
<thead>
<tr>
<th>Degree name</th>
<th>H5Fclose behavior with no open object in file</th>
<th>H5Fclose behavior with open object(s) in file</th>
</tr>
</thead>
<tbody>
<tr>
<td>H5F_CLOSE_WEAK</td>
<td>Actual file is closed.</td>
<td>Access to file identifier is terminated; actual file close is delayed until all objects in file are closed</td>
</tr>
<tr>
<td>H5F_CLOSE_SEMI</td>
<td>Actual file is closed.</td>
<td>Function returns FAILURE</td>
</tr>
<tr>
<td>H5F_CLOSE_STRONG</td>
<td>Actual file is closed.</td>
<td>All open objects remaining in the file are closed then file is closed</td>
</tr>
<tr>
<td>H5F_CLOSE_DEFAULT</td>
<td>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.</td>
<td></td>
</tr>
</tbody>
</table>

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

```fortran
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:**

```c
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

```fortran
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:**  
```c
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:** h5set_fill_value_f
```fortran
SUBROUTINE h5set_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 h5set_fill_value_f
```
**Name:** H5Pset_filter  
**Signature:**
```c
herr_t H5Pset_filter(hid_t plist, H5Z_filter_t filter, 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 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` 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.

Parameters:

- `hid_t plist` IN: Property list identifier.
- `H5Z_filter_t filter` 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`

```fortran
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 listplist.

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, void *buf, size_t buf_size, void *op_data)

where filter 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:**

```c
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

```fortran
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
END SUBROUTINE h5pset_fletcher32_f
```

**History:**
- **Release C**
  - 1.6.0 Function introduced in this release.
Name: H5Pset_gc_references

Signature:

\[
\text{herr_t H5Pset_gc_reference(hid_t \text{plist, unsigned gc_ref})}
\]

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:

\[
\begin{align*}
\text{hid_t \text{plist}} & \quad \text{IN: File access property list identifier.} \\
\text{unsigned gc_ref} & \quad \text{IN: Flag setting reference garbage collection to on (1) or off (0).}
\end{align*}
\]

Returns:
Returns a non−negative value if successful; otherwise returns a negative value.

Fortran90 Interface: h5pset_symlinks_f

SUBROUTINE h5pset_symlinks_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
\text{references for the file}
INTEGER, INTENT(OUT) :: hdferr ! Error code
\text{0 on success and −1 on failure}
END SUBROUTINE h5pset_symlinks_f
Name: H5Pset_hyper_cache

Signature:

    herr_t H5Pset_hyper_cache(hid_t plist, unsigned cache, unsigned limit)

Purpose:

Indicates whether to cache hyperslab blocks during I/O.

Description:

[NOTE: This function is deprecated in HDF5 Release 1.6 and will eventually be removed from the HDF5 distribution. It is provided in this release only to enable backward compatibility with HDF5 Releases 1.4.x and is enabled only if the HDF5 library is configured with the flag H5_WANT_H5_V1_4_COMPAT; the function is not enabled in the binaries distributed by NCSA. ]

Given a dataset transfer property list, H5Pset_hyper_cache indicates whether to cache hyperslab blocks during I/O, a process which can significantly increase I/O speeds.

When working with hyperslab selections, it is possible to significantly speed up I/O operations by retrieving an entire hyperslab from the file in one operation and caching it in memory. The cache parameter specifies whether to turn caching on for hyperslab I/O operations. If cache is set to 1, caching is turned on; if set to 0, caching is turned off.

The parameter limit sets the maximum size of the hyperslab block to cache. If a block is smaller than that limit, it may still not be cached if no memory is available. Setting limit to 0 (zero) indicates no limitation on the size of block to attempt to cache.

The default is to cache blocks with no limit on block size for serial I/O and to not cache blocks for parallel I/O.

Parameters:

    hid_t plist
       IN: Dataset transfer property list identifier.

    unsigned cache
       IN: A flag indicating whether caching is to be set to on (1) or off (0).

    unsigned limit
       IN: Maximum size of the hyperslab block to cache. 0 (zero) indicates no limit.

Returns:

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

Fortran90 Interface: h5pset_hyper_cache_f

SUBROUTINE h5pset_hyper_cache_f(prp_id, cache, limit, hdferr)
IMPLICIT NONE
INTEGER(HID_T), INTENT(IN) :: prp_id ! Property list identifier
INTEGER, INTENT(IN) :: cache       !
INTEGER, INTENT(IN) :: limit        ! Maximum size of the hyperslab block to cache
INTEGER, INTENT(OUT) :: hdferr      ! Error code
                        ! 0 on success and -1 on failure
END SUBROUTINE h5pset_hyper_cache_f
Name: H5Pset_hyper_vector_size

Signature:

\[ \text{herr}_t \text{H5Pset_hyper_vector_size} (\text{hid}_t \text{dxpl_id}, \text{size}_t \text{vector_size}) \]

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:

- \text{hid}_t \text{dxpl_id} \quad \text{IN: Dataset transfer property list identifier.}
- \text{size}_t \text{vector_size} \quad \text{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:
-plist
  IN: Identifier of property list to query.

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_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:

```
err_t H5Pset_multi_type(hid_t fapl_id, H5FD_mem_t type)
```

Purpose:
Sets data type property for MULTI driver.

Description:
H5Pset_multi_type sets the data type property in 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 specified in type may be one of the following:

- H5FD_MEM_DEFAULT: Need description....
- H5FD_MEM_SUPER: Super block ... need description....
- H5FD_MEM_BTREE: Btree ... need description....
- H5FD_MEM_DRAW: Need description....
- H5FD_MEM_GHEAP: Global heap ... need description....
- H5FD_MEM_LHEAP: Local Heap ... need description....
- H5FD_MEM_OHDR: Need description....

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:** H5Pset_preserve  

**Signature:**

```c
herr_t H5Pset_preserve(hid_t plist, hbool_t status)
```

**Purpose:**

Sets the dataset transfer property list status to TRUE or FALSE.

**Description:**

H5Pset_preserve sets the dataset transfer property list status to TRUE or FALSE.

When reading or writing compound data types 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

```fortran
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_shuffle

Signature:

```c
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

```fortran
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:**

```c
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 IO 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 IO 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

```fortran
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:**

```c
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: Identifier for property list to query.
- `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

```fortran
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 up for a dataset containing a non-allowed datatype, H5Pset_szip will succeed but the subsequent call to H5Dcreate will fail; the conflict is detected only when the property list is used.

SZIP options are passed in an options mask, options_mask, as follows.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>H5_SZIP_EC_OPTION_MASK</td>
<td>Selects entropy coding method.</td>
</tr>
<tr>
<td>H5_SZIP_NN_OPTION_MASK</td>
<td>Selects nearest neighbor coding method.</td>
</tr>
</tbody>
</table>

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 applies the EC method as above.

Other factors may affect results, but the above criteria provide 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 is 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.

`H5Pset_szip` will fail if SZIP encoding is disabled in the available copy of the SZIP library. `H5Zget_filter_info` can be employed to avoid such a failure.

**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:**
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 \texttt{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 \texttt{H5Tset\_offset} or \texttt{H5Tget\_offset}), the data is in lowest \( N \) bits of the data element. In this case, the SZIP \texttt{bits\_per\_pixel} is set to the precision of the HDF5 datatype.
◊ If the offset is not zero, the SZIP \texttt{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 \texttt{bits\_per\_pixel} will be set to 32.
◊ If the HDF5 datatype precision is 33-bit to 64-bit, the SZIP \texttt{bits\_per\_pixel} will be set to 64.

HDF5 always modifies the options mask provided by the user to set up usage of \texttt{RAW\_OPTION\_MASK}, \texttt{ALLOW\_K13\_OPTION\_MASK}, and one of \texttt{LSB\_OPTION\_MASK} or \texttt{MSB\_OPTION\_MASK}, depending on endianness of the datatype.

\textbf{Fortran90 Interface: h5pset\_szip\_f}

\begin{verbatim}
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
\end{verbatim}

\textbf{History:}

\textbf{Release C}

1.6.0 Function introduced in this release.
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:**
```c
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:
```c
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.

The C Interfaces:

- H5Rcreate
- H5Rdereference
- H5Rget_region
- H5Rget_obj_type

Alphabetical Listing

- H5Rcreate
- H5Rdereference
- H5Rget_obj_type
- H5Rget_object_type
- H5Rget_region

* Functions labelled with an asterisk (*) are provided only for backwards compatibility with HDF5 Releases 1.4.x. See further notes in the description of each function.

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
Name: H5Rcreate

Signature:

```c
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.

<table>
<thead>
<tr>
<th>hdset_reg_ref_t</th>
<th>H5R_DATASET_REGION</th>
<th>Dataset region reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>hobj_ref_t</td>
<td>H5R_OBJECT</td>
<td>Object reference</td>
</tr>
</tbody>
</table>

The parameters `loc_id` and `name` are used to locate the object.

The parameter `space_id` identifies the region to be pointed to for a dataset region reference. This parameter is unused with object references.

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 for dataset region references.

Returns:

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

Fortran90 Interface: h5rcreate_f

To create an object reference

```fortran
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
  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
END SUBROUTINE h5rcreate_f
**Name:** H5Rdereference

**Signature:**

```c
hid_t H5Rdereference(hid_t dataset, H5R_type_t ref_type, void *ref)
```

**Purpose:**

Opens the HDF5 object referenced.

**Description:**

Given a reference to some object, H5Rdereference opens that object and returns an identifier.

The parameter `ref_type` specifies the reference type of `ref`. `ref_type` may contain either of the following values:

- H5R_OBJECT (0)
- H5R_DATASET_REGION (1)

**Parameters:**

- `hid_t dataset` IN: Dataset containing reference object.
- `H5R_type_t ref_type` IN: The reference type of `ref`.
- `void *ref` IN: Reference to open.

**Returns:**

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

**Fortran90 Interface:** `h5rdereference_f`

**To dereference an object**

```fortran
SUBROUTINE h5rdereference_f(dset_id, ref, obj_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id ! Dataset identifier
  TYPE(hobj_ref_t_f), INTENT(IN) :: ref ! Object reference
  INTEGER(HID_T), INTENT(OUT) :: obj_id ! Object identifier
  INTEGER, INTENT(OUT) :: hdferr ! Error code
END SUBROUTINE h5rdereference_f
```

**To dereference a region**

```fortran
SUBROUTINE h5rdereference_f(dset_id, ref, obj_id, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: dset_id ! Dataset identifier
  TYPE(hdset_reg_ref_t_f), INTENT(IN) :: ref ! Object reference
  INTEGER(HID_T), INTENT(OUT) :: obj_id ! Object identifier
  INTEGER, INTENT(OUT) :: hdferr ! Error code
END SUBROUTINE h5rdereference_f
```
Name: H5Rget_obj_type

Signature:

```c
H5G_obj_t H5Rget_obj_type(hid_t id, H5R_type_t ref_type, void *ref)
```

Purpose:

Retrieves the type of object that an object reference points to.

Description:

Given type of object reference, ref_type, and a reference to an object, ref, H5Rget_obj_type returns the type of the referenced object.

Valid object reference types, to pass in as ref_type, include the following:

- **H5R_OBJECT**: Reference is an object reference.
- **H5R_DATASET_REGION**: Reference is a dataset region reference.

Valid object type return values include the following:

- **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 id**: IN: The dataset containing the reference object or the location identifier of the object that the dataset is located within.
- **H5R_type_t ref_type**: IN: Type of reference to query.
- **void *ref**: IN: Reference to query.

Returns:

Returns an object type as defined in H5Gpublic.h if successful; otherwise returns H5G_UNKNOWN.

Fortran90 Interface: h5rget_object_type_f

```fortran
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
END SUBROUTINE h5rget_object_type_f
```
**Name:** H5Rget_object_type  
**Signature:**

```c
int H5Rget_object_type(hid_t id, void *ref )
```

**Purpose:**
Retrieves the type of object that an object reference points to.

**Description:**

[NOTE: This function is provided only to enable backward compatibility with HDF5 Releases 1.4.x. This function is enabled only if the HDF5 library is configured with the flag H5_WANT_H5_V1_4_COMPAT and is not enabled in the binaries distributed by NCSA. This function has been replaced in Release 1.6 by the function H5Rget_obj_type and will eventually be deleted from the HDF5 distribution.]

Given a reference to an object ref, H5Rget_object_type returns the type of the object pointed to.

**Parameters:**

- `hid_t id`,  
  *IN:* The dataset containing the reference object or the location identifier of the object that the dataset is located within.

- `void *ref`,  
  *IN:* Reference to query.

**Returns:**
Returns an object type as defined in H5Gpublic.h; otherwise returns H5G_UNKNOWN.

**History:**

- **Release C**  
  1.6.0 Function introduced in this release.
Name: H5Rget_region

Signature:

hid_t H5Rget_region(hid_t dataset, H5R_type_t ref_type, void *ref)

Purpose:

Retrieves a dataspace with the specified region selected.

Description:

Given a reference to an object ref, H5Rget_region creates a copy of the dataspace of the dataset pointed to and defines a selection in the copy which is the region pointed to.

The parameter ref_type specifies the reference type of ref. ref_type may contain the following value:

◊ H5R_DATASET_REGION (1)

Parameters:

hid_t dataset IN: Dataset containing reference object.
H5R_type_t ref_type IN: The reference type of ref.
void *ref IN: Reference to open.

Returns:

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

Fortran90 Interface: h5rget_region_f

SUBROUTINE h5rget_region_f(dset_id, ref, space_id, hdferr)
    IMPLICIT NONE
    INTEGER(HID_T), INTENT(IN) :: dset_id ! Dataset 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

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
- H5Screate_simple
- H5Sis_simple
- H5Soffset_simple
- H5Sget_simple_extent_dims
- H5Sget_simple_extent_ndims
- H5Sget_simple_extent_npoints
- H5Sget_simple_extent_type
- H5Sextent_copy
- 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
- H5Sget_select_hyper_nblocks
- H5Sget_select_hyper_blocklist
- H5Sget_select_hyper_nblocks
- H5Sget_select_hyper_blocklist
- H5Sget_select_bounds
- H5Sget_select_hyper_nblocks
- H5Sget_select_hyper_blocklist
- H5Sget_select_elem_npoints
- H5Sget_select_elem_pointlist
- H5Sget_select_type
- H5Sget_select_extent_type
- H5Sget_extent_type
- H5Sget_extent_simple
- H5Sget_extent_none

Alphabetical Listing

- H5Sclose
- H5Scopy
- H5Screate
- H5Screate_simple
- H5Sextent_copy
- H5Sget_select_bounds
- H5Sget_select_elem_npoints
- H5Sget_select_elem_pointlist
- H5Sget_select_hyper_blocklist
- H5Sget_select_hyper_nblocks
- H5Sget_select_hyper_blocklist
- H5Sget_select_all
- H5Sselect_elements
- H5Sselect_hyperslab
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_elem_npoints
- H5Sselect_elem_pointlist
- H5Sselect_bounds
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_all
- H5Sselect_elements
- H5Sselect_hyperslab
- H5Sselect_elem_npoints
- H5Sselect_elem_pointlist
- H5Sselect_bounds
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_all
- H5Sselect_elem_npoints
- H5Sselect_elem_pointlist
- H5Sselect_bounds
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_all
- H5Sselect_elem_npoints
- H5Sselect_elem_pointlist
- H5Sselect_bounds
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_all
- H5Sselect_elem_npoints
- H5Sselect_elem_pointlist
- H5Sselect_bounds
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_all
- H5Sselect_elem_npoints
- H5Sselect_elem_pointlist
- H5Sselect_bounds
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist
- H5Sselect_hyper_nblocks
- H5Sselect_hyper_blocklist

The FORTRAN90 Interfaces:

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

- h5screate_f
- h5scopy_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
- 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
- h5sset_elements_f
- h5sselect_all_f
- h5sselect_none_f
- h5sselect_valid_f
- h5sselect_hyperslab_f
**Name:** H5Sclose  
**Signature:**  
\[
herr_t \text{H5Sclose}(hid_t \text{space}_id)  
\]
**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:**  
\[
\begin{align*}
\text{hid}_t \text{space}_id & \quad \text{Identifier of dataspace to release.} 
\end{align*}
\]

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

**Fortran90 Interface:** h5sclose_f  
```fortran
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
**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. The types currently supported are H5S_SCALAR and H5S_SIMPLE; others are planned to be added later.

**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

```fortran
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
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
```
**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.

`rank` is the number of dimensions used in the dataspace.

`dims` is an array specifying the size of each dimension of the dataset while `maxdims` is an array 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, (-1), the maximum size of the corresponding dimension is unlimited. Otherwise, no element of `maxdims` should be smaller than the corresponding element of `dims`.

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.

**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: H5Sextent_copy

Signature:

```c
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

```fortran
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: H5Sget_select_bounds
Signature:
\[ \text{herr_t H5Sget_select_bounds(hid_t space_id, hsize_t *start, hsize_t *end) } \]

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 \texttt{start} and \texttt{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 \texttt{FAIL}.

Parameters:
- \texttt{hid_t space_id} IN: Identifier of dataspace to query.
- \texttt{hsize_t *start} OUT: Starting coordinates of the bounding box.
- \texttt{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 \texttt{start} and \texttt{end} parameters have changed from type \texttt{hsize_t *} to \texttt{hssize_t *}. 

260
**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

```fortran
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 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
INTEGER, INTENT(OUT) :: hdferr             ! Error code
END SUBROUTINE h5sget_select_elem_pointlist_f
```
**Name:** H5Sget_select_hyper_blocklist

**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

```fortran
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
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:

\[
\text{H5S\_sel\_type} \ \text{H5Sget\_select\_type}(\text{hid\_t space\_id})
\]

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:
- \text{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:

- \text{H5S\_SEL\_NONE}
  No selection is defined.
- \text{H5S\_SEL\_POINTS}
  A sequence of points is selected.
- \text{H5S\_SEL\_HYPERSLABS}
  A hyperslab or compound hyperslab is selected.
- \text{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:**

```c
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

```fortran
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
  ! Data space 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:

\texttt{hssize_t H5Sget_simple_extent_npoints(hid_t space_id)}

Purpose:

Determines the number of elements in a dataspace.

Description:

\texttt{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:

\texttt{hid_t space_id} \quad \text{ID of the dataspace object to query}

Returns:

Returns the number of elements in the dataspace if successful; otherwise returns 0.

Fortran90 Interface: \texttt{h5sget_simple_extent_npoints_f}

\begin{verbatim}
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
\end{verbatim}

Name: H5Sget_simple_extent_type

Signature:

\[ \text{H5S_class_t H5Sget_simple_extent_type}(\text{hid_t space_id}) \]

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:**

```c
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

```fortran
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
**Name:** H5Sselect_all  

**Signature:**

\[
\text{herr_t H5Sselect_all(hid_t space_id)}
\]

**Purpose:**  
S\selects the entire dataspace.

**Description:**

H5Sselect_all selects the entire extent of the dataspace `space_id`.

More specifically, H5Sselect_all selects the special \text{5S\_SELECT\_ALL} region for the dataspace `space_id`. H5S\_SELECT\_ALL selects the entire dataspace for any dataspace it is applied to.

**Parameters:**

- `hid_t space_id`  
  IN: The identifier for the dataspace in which the selection is being made.

**Returns:**

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

**Fortran90 Interface:**

```
SUBROUTINE h5sselect_all_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_all_f
```
**Name:** H5Sselect_elements

**Signature:**
```
herr_t H5Sselect_elements(hid_t space_id, H5S_seloper_t op, const 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 dataspace_rank by num_elements 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.
- `const 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`

```fortran
SUBROUTINE h5sselect_elements_f(space_id, operator, 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_OR_F
INTEGER, INTENT(IN) :: num_elements ! Number of elements to be selected
INTEGER(HSIZE_T), DIMENSION(*,*) :: coord ! Array with the coordinates
                                ! of the selected elements:
                                ! coord(num_elements, rank)
```

274
INTEGER, INTENT(OUT) :: hdferr

! Error code
! 0 on success and -1 on failure

END SUBROUTINE h5sselect_elements_f

**History:**

**Release C**

1.6.4  coord parameter type changed to const hsize_t.

**Fortran90**

coord parameter type changed to INTEGER(HSIZE_T).
**Name:** H5Sselect_hyperslab  

**Signature:** 

\[ 
\text{herr}_t \text{H5Sselect_hyperslab}(\text{hid}_t \text{ space}_id, \text{H5S seloper}_t \text{ op}, \text{const hsize}_t \text{*start}, \text{const hsize}_t \text{*stride}, \text{const hsize}_t \text{*count}, \text{const hsize}_t \text{*block}) 
\]

**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.

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 determines the starting coordinates of the hyperslab to select.

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 an array 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 all 1's).

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 the `block` array were set to all 1's).
For example, in a 2−dimensional dataspace, setting start to \([1,1]\), stride to \([4,4]\), count to \([3,7]\), and block to \([2,2]\) selects 21 2x2 blocks of array elements starting with location \((1,1)\) and selecting blocks at locations \((1,1)\), \((5,1)\), \((9,1)\), \((1,5)\), \((5,5)\), 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`

```fortran
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  
            ! Starting coordinates 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**
  - 1.6.4  
    start[] parameter type changed to `const hsize_t`.

- **Fortran90**
  - 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

```fortran
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:

\[
\text{herr_t } \text{H5Sset_extent_none}(\text{hid_t } \text{space_id})
\]

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:
\[
\text{hid_t } \text{space_id} \quad \text{The identifier for the dataspace from which the extent is to be removed.}
\]

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` Dataspac 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 dataspace identifier 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) :: maximum_size
   ! 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.

The C Interfaces:

General Datatype Operations
- H5Tcreate
- H5Topen
- H5Tcommit
- H5Tcommitted
- H5Tcopy
- H5Tequal
- H5Tlock
- H5Tget_class
- H5Tget_size
- H5Tget_super
- H5Tdetect_class
- H5Tclose

Atomic Datatype Properties
- H5Tset_size
- H5Tset_order
- H5Tset_order
- H5Tget_precision
- H5Tset_precision
- H5Tget_offset
- H5Tset_offset
- H5Tget_pad
- H5Tset_pad
- H5Tget_sign
- H5Tset_sign
- H5Tget_fields
- H5Tget_fields
- H5Tget_ebias
- H5Tset_ebias
- H5Tget_norm
- H5Tset_norm
- H5Tget_inpad
- H5Tset_inpad
- H5Tget_cpuset
- H5Tset_cpuset
- H5Tget_strpad
- H5Tset_strpad

Conversion Functions
- H5Tconvert
- H5Tfind
- H5Tset_overflow
- H5Tget_overflow
- H5TRegister
- H5Tunregister

Enumeration Datatypes
- H5Tenum_create
- H5Tenum_insert
- H5Tenum_nameof
- H5Tenum_valueof
- H5Tget_member_value
- H5Tget_nmembers
- H5Tget_member_name
- H5Tget_member_index

Compound Datatype Properties
- H5Tget_nmembers
- H5Tget_member_class
- H5Tget_member_name
- H5Tget_member_index
- H5Tget_member_offset
- H5Tget_member_type
- H5Tinsert
- H5Tpack

Array Datatypes
- H5Tarray_create
- H5Tget_array_ndims
- H5Tget_array_dims

Variable-length Datatypes
- H5Tvlen_create
- H5Tis_variable_str

Opaque Datatypes
- H5Tset_tag
- H5Tget_tag
HDF5 Reference Manual

Alphabetical Listing

- H5Tarray_create
- H5Tclose
- H5Tcommit
- H5Tcommitted
- H5Tconvert
- H5Tcopy
- H5Tcreate
- H5Tdetect_class
- H5Tenum_create
- H5Tenum_insert
- H5Tenum_nameof
- H5Tenum_valueof
- H5Tequal
- H5Tfind
- H5Tget_array_dims
- H5Tget_array_ndims
- H5Tget_class
- 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_overflow
- H5Tget_pad
- H5Tget_precision
- H5Tget_sign
- H5Tget_size
- H5Tget_strpad
- H5Tget_super
- H5Tget_tag
- H5Tinsert
- H5Tis_variable_str
- H5Tlock
- H5Topen
- H5Tpack
- H5Tregister
- H5Tset_cset
- H5Tset_ebias
- H5Tset_fields
- H5Tset_inpad
- H5Tset_norm
- H5Tset_offset
- H5Tset_order
- H5Tset_overflow
- 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
- h5topen_f
- h5tcommit_f
- h5tcommitted_f
- h5tcopy_f
- h5tequal_f
- h5tclose_f
- h5tget_class_f
- h5tget_size_f
- h5tget_super_f
- h5tget_inpad_f

Enumeration Datatypes
- h5tenum_create_f
- h5tenum_insert_f
- h5tenum_nameof_f
- h5tenum_valueof_f
- h5tequal_f
- h5tget_class_f
- h5tget_size_f
- h5tget_super_f
- h5tget_tag_f
- h5tinsert_f
- h5tis_variable_str_f

Atomic Datatype Properties
- h5tset_size_f
- h5tget_precision_f
- h5tset_precision_f
- h5tget_order_f
- h5tset_order_f
- h5tset_order_f
- h5tget_order_f
- h5tset_order_f
- h5tget_fields_f
- h5tset_fields_f
- h5tget_ebias_f
- h5tset_ebias_f
- h5tget_ebias_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
- h5tget_member_value_f
- h5tget_native_type_f
- h5tget_nmembers_f
- h5tget_norm_f
- h5tget_offset_f
- h5tget_order_f
- h5tget_overflow_f
- h5tget_pad_f
- h5tget_precision_f
- h5tget_sign_f
- h5tget_size_f
- h5tget_strpad_f
- h5tget_super_f
- h5tget_tag_f
- h5tinsert_f
- h5tis_variable_str_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

Signature:

hid_t H5Tarray_create( hid_t base, int rank, const hsize_t dims[/*rank*/), const int perm[/*rank*/),

Purpose:

Creates an array datatype object.

Description:

H5Tarray_create creates a new array datatype object.

base 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 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 non-negative value 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.
**Name:** H5Tclose

**Signature:**

```c
herr_t H5Tclose(hid_t type_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 type_id` Identifier of datatype to release.

**Returns:**

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

**Fortran90 Interface:** h5tclose_f

```fortran
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 type)
```

**Purpose:**
Commits a transient datatype to a file, creating a new named datatype.

**Description:**
H5Tcommit commits a transient datatype (not immutable) to a file, turned it into a named datatype. The `loc_id` is either a file or group identifier which, when combined with `name`, refers to a new named datatype.

**Parameters:**
- `hid_t loc_id` **IN:** A file or group identifier.
- `const char * name` **IN:** A datatype name.
- `hid_t type` **IN:** A datatype identifier.

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

**Fortran90 Interface:** `h5tcommit_f`

```fortran
SUBROUTINE h5tcommit_f(loc_id, name, type_id, hdferr)
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
END SUBROUTINE h5tcommit_f
```
Name: H5Tcommitted

Signature:

htri_tH5Tcommitted(hid_t type)

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 type 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 type             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(type_id, hdferr)
IMPLICIT NONE
INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
INTEGER, INTENT(OUT) :: hdferr       ! Error code
END SUBROUTINE h5tcommitted_f
Name: H5Tconvert

Signature:

```c
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 type_id)`

**Purpose:**
Copies an existing datatype.

**Description:**
H5Tcopy copies an existing datatype. The returned type is always transient and unlocked.

The `type_id` argument can be either a datatype identifier, a predefined datatype (defined in H5Tpublic.h), or a dataset identifier. If `type_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 type_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

```fortran
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
  INTEGER(HID_T), INTENT(OUT) :: type_id     ! Datatype identifier
  INTEGER, INTENT(OUT) :: hdferr             ! Error code
  ! H5T_COMPOUND_F (6)
  ! H5T_ENUM_F     (8)
  ! H5T_OPAQUE_F   (9)

  INTEGER(SIZE_T), INTENT(IN) :: size        ! Size of the datatype

END SUBROUTINE h5tcreate_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: H5Tenum_create

**Signature:**

\[ hid_t \text{H5Tenum\_create}(hid_t \text{parent\_id}) \]

**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:** h5t_enum_create_f

```fortran
SUBROUTINE h5t_enum_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 h5t_enum_create_f
```
**Name:** H5Tenum_insert  

**Signature:**  

```c
herr_t H5Tenum_insert(hid_t type, 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.

- `type` 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 `type`.

- `value` points to data which is of the datatype defined when the enumeration datatype was created.

**Parameters:**  

- `hid_t type` - 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  

```fortran
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:**

```c
herr_t H5Tenum_nameof(hid_t type, 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 type.

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 type`: 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

```fortran
SUBROUTINE h5tenum_nameof_f(type_id, name, namelen, value, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id  ! Datatype identifier
  CHARACTER(LEN=*) , INTENT(OUT) :: name  ! Name of the enumeration datatype
  INTEGER(SIZE_T), INTENT(IN) :: namelen ! Length of the name
  INTEGER, INTENT(IN) :: value          ! Value of the enumeration datatype
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tenum_nameof_f
```
Name: H5Tenum_valueof

Signature:

```c
herr_t H5Tenum_valueof(hid_t type, 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 type.

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 type` 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

```fortran
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:**

\[ htri_t \text{H5Tequal}(hid_t \text{type_id1}, hid_t \text{type_id2}) \]

**Purpose:**

Determines whether two datatype identifiers refer to the same datatype.

**Description:**

H5Tequal determines whether two datatype identifiers refer to the same datatype.

**Parameters:**

- \( hid_t \text{type_id1} \) Identifier of datatype to compare.
- \( hid_t \text{type_id2} \) 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

```fortran
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:**

```c
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

**Signature:**

```c
int H5Tget_array_dims(hid_t adtype_id, hsize_t *dims[], int *perm[])
```

**Purpose:**
Retrieves sizes of array dimensions and dimension permutations.

**Description:**

H5Tget_array_dims 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 dimension permutations, i.e., C versus FORTRAN array order, are returned in the array `perm`.

**Parameters:**

- `hid_t adtype_id` IN: Datatype identifier of array object.
- `hsize_t *dims[]` OUT: Sizes of array dimensions.
- `int *perm[]` OUT: Dimension permutations.

**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

```fortran
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(*) :: 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.
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 type_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_TIME
◊ 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.

**Parameters:**

*hid_t type_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_TIME_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_cset

**Signature:**

\[ \text{H5T}_\text{cset}_\text{t} \text{ H5Tget_cset(hid}_t \text{ type}_\text{id}) \]

**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:

\[ \text{H5T}_\text{CSET}_\text{ASCII} (0) \]

Character set is US ASCII

**Parameters:**

- \( \text{hid}_t \text{ type}_\text{id} \) Identifier of datatype to query.

**Returns:**

Returns a valid character set type if successful; otherwise H5T_CSET_ERROR (-1).

**Fortran90 Interface:** h5tget_cset_f

```fortran
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 = 0
INTEGER, INTENT(OUT) :: hdferr ! Error code
END SUBROUTINE h5tget_cset_f
```
**Name:** H5Tget_ebias  
**Signature:**
```
size_t H5Tget_ebias(hid_t type_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 type_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, INTENT(OUT) :: ebias         ! Datatype exponent bias
  INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tget_ebias_f
```
Name: H5Tget_fields

Signature:

```c
herr_t H5Tget_fields(hid_t type_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 type_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

```fortran
SUBROUTINE h5tget_fields_f(type_id, epos, esize, mpos, msize, hdferr)
IMPLICIT NONE
INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
INTEGER, INTENT(OUT) :: epos        ! Exponent bit-position
INTEGER, INTENT(OUT) :: esize       ! Size of exponent in bits
INTEGER, INTENT(OUT) :: mpos        ! Mantissa bit-position
INTEGER, INTENT(OUT) :: msize       ! Size of mantissa in bits
INTEGER, INTENT(OUT) :: hdferr      ! Error code
END SUBROUTINE h5tget_fields_f
```
Name: H5Tget_inpad

Signature:

\[ H5T\_pad\_t \text{H5Tget_inpad}(hid\_t\ type\_id) \]

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:

- \( hid\_t\ type\_id \) 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:

\[ \text{H5T_class_t H5Tget_member_class( hid_t cdtype_id, unsigned member_no )} \]

Purpose:
Returns datatype class of compound datatype member.

Description:
Given a compound datatype, \( cdtype_id \), the function \( \text{H5Tget_member_class} \) returns the datatype class of the compound datatype member specified by \( \text{member_no} \).

Parameters:
- \( \text{hid_t cdtype_id} \) IN: Datatype identifier of compound object.
- \( \text{unsigned member_no} \) 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

```fortran
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 \textit{unsigned}.  

308
Name: H5Tget_member_index

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 type_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  
<table>
<thead>
<tr>
<th><strong>Fortran90</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td>1.4.5</td>
</tr>
<tr>
<td>Function introduced in this release.</td>
</tr>
<tr>
<td>1.4.4</td>
</tr>
<tr>
<td>Function introduced in this release.</td>
</tr>
</tbody>
</table>

HDF5 Reference Manual H5Tget_member_index
**Name:** H5Tget_member_name  
**Signature:**
```c
char * H5Tget_member_name(hid_t type_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 type_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
```fortran
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  membro parameter type changed to `unsigned`.
Name: H5Tget_member_offset

Signature:

\[
\text{size_t H5Tget_member_offset(hid_t type_id, unsigned memb_no)}
\]

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:
- \text{hid_t type_id} Identifier of datatype to query.
- \text{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
INTEGER(SIZE_T), INTENT(OUT) :: offset ! Byte offset of the the
INTEGER, INTENT(OUT) :: hdferr ! Beginning of the field
! 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 type_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 type_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 type, 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 type       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 member
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 type_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 type_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.

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. Time, bifield, opaque, and reference datatypes are returned as a copy of type_id.

The identifier returned by H5Tget_native_type should eventually be closed by calling H5Tclose to release resources.

Parameters:

hid_t type_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:
None.

History:
Release C
1.6.0 Function introduced in this release.
Name: H5Tget_nmembers  
Signature:  
    int H5Tget_nmembers(hid_t type_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 type_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 type_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 type_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 type_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):

<table>
<thead>
<tr>
<th>Byte Position</th>
<th>Big−Endian Offset=0</th>
<th>Big−Endian Offset=16</th>
<th>Little−Endian Offset=0</th>
<th>Little−Endian Offset=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>[ pad]</td>
<td>[0x11]</td>
<td>[0x22]</td>
<td>[ pad]</td>
</tr>
<tr>
<td>1:</td>
<td>[ pad]</td>
<td>[0x22]</td>
<td>[0x11]</td>
<td>[ pad]</td>
</tr>
<tr>
<td>2:</td>
<td>[0x11]</td>
<td>[ pad]</td>
<td>[ pad]</td>
<td>[0x22]</td>
</tr>
<tr>
<td>3:</td>
<td>[0x22]</td>
<td>[ pad]</td>
<td>[ pad]</td>
<td>[0x11]</td>
</tr>
</tbody>
</table>

Parameters:

    hid_t type_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, 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:**

\[
H5T_order_t \text{H5Tget_order(hid_t type_id)}
\]

**Purpose:**

Returns the byte order of an atomic datatype.

**Description:**

H5Tget_order returns the byte order of an atomic datatype.

Possible return values are:

\[
H5T\_ORDER\_LE(0) \\
\text{Little endian byte ordering (default).}
\]

\[
H5T\_ORDER\_BE(1) \\
\text{Big endian byte ordering.}
\]

\[
H5T\_ORDER\_VAX(2) \\
\text{VAX mixed byte ordering (not currently supported).}
\]

**Parameters:**

hid_t type_id 
Identifier of datatype to query.

**Returns:**

Returns a byte order constant if successful; otherwise H5T_ORDER_ERROR (-1).

**Fortran90 Interface:** h5tget_order_f

```fortran
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_overflow

Signature: H5Tget_overflow ()

    H5Toverflow_t H5Tget_overflow(void)

Purpose:
    Returns a pointer to the current global overflow function.

Description:
    H5Tset_overflow returns a pointer to the current global overflow function. This is an
    application–defined function that is called whenever a datatype conversion causes an overflow.

Parameters:
    None.

Returns:
    Returns a pointer to an application–defined function if successful. Otherwise returns NULL; this can
    happen if no overflow handling function is registered.

Fortran90 Interface:
    None.
Name: H5Tget_pad

Signature:

herr_t H5Tget_pad(hid_t type_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 type_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

```fortran
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:**

```c
size_t H5Tget_precision(hid_t type_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 type_id` Identifier of datatype to query.

**Returns:**

Returns the number of significant bits if successful; otherwise 0.

**Fortran90 Interface:** h5tget_precision_f

```fortran
SUBROUTINE h5tget_precision_f(type_id, precision, hdferr)
    IMPLICIT NONE
    INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
    INTEGER, 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 type_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 type_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:**

\[
\text{size_t H5Tget_size(hid_t type_id)}
\]

**Purpose:**

Returns the size of a datatype.

**Description:**

H5Tget_size returns the size of a datatype in bytes.

**Parameters:**

- \textit{hid_t type_id}  
  Identifier of datatype to query.

**Returns:**

Returns the size of the datatype in bytes if successful; otherwise 0.

**Fortran90 Interface:** h5tget_size_f

```fortran
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:

\[ H5T_{\text{str}} \_t \text{H5Tget\_strpad}(\text{hid} \_t \text{type}\_id) \]

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 type_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

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
END SUBROUTINE h5tget_strpad_f
Name: H5Tget_super

Signature:

hid_t H5Tget_super(hid_t type)

Purpose:

Returns the base datatype from which a datatype is derived.

Description:

H5Tget_super returns the base datatype from which the datatype type is derived.

In the case of an enumeration type, the return value is an integer type.

Parameters:

hid_t type 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) :: base_type_id ! Base datatype identifier
    INTEGER, INTENT(OUT) :: hdferr ! Error code
END SUBROUTINE h5tget_super_f
Name: H5Tget_tag

Signature:

char *H5Tget_tag(hid_t type_id)

Purpose:
Gets the tag associated with an opaque datatype.

Description:
H5Tget_tag returns the tag associated with the opaque datatype type_id.

The tag is returned via a pointer to an allocated string, which the caller must free.

Parameters:
hid_t type_id Datatype identifier for the opaque datatype.

Returns:
Returns a pointer to an allocated string if successful; otherwise returns NULL.

Fortran 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
END SUBROUTINE h5tget_tag_f
Name: H5Tinsert

Signature:

    herr_t H5Tinsert(hid_t type_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 type_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 type_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

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:
H5Tvlen_create 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
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 type_id)

Purpose:

Locks a datatype.

Description:

H5Tlock locks the datatype specified by the type_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 type_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)

Purpose:

Opens a named datatype.

Description:

H5Topen 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: h5topen_f

SUBROUTINE h5topen_f(loc_id, name, type_id, hdferr)
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
END SUBROUTINE h5topen_f
Name: H5Tpack

Signature:

    herr_t H5Tpack(hid_t type_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 type_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:**

```c
herr_t H5Tset_cset(hid_t type_id, H5T_cset_t cset)
```

**Purpose:**

Sets character set to be used.

**Description:**

H5Tset_cset 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 type_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

```fortran
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 type_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 type_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`

```fortran
SUBROUTINE h5tset_ebias_f(type_id, ebias, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, 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:

```c
herr_t H5Tset_fields(hid_t type_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 type_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

```fortran
SUBROUTINE h5tset_fields_f(type_id, epos, esize, mpos, msize, hdferr)
    IMPLICIT NONE
    INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
    INTEGER, INTENT(IN) :: epos           ! Exponent bit−position
    INTEGER, INTENT(IN) :: esize          ! Size of exponent in bits
    INTEGER, INTENT(IN) :: mpos           ! Mantissa bit−position
    INTEGER, INTENT(IN) :: msize          ! Size of mantissa in bits
    INTEGER, INTENT(OUT) :: hdferr        ! Error code
END SUBROUTINE h5tset_fields_f
```
Name: H5Tset_inpad

Signature:

\[ \text{herr\_t H5Tset\_inpad(hid\_t type\_id, H5T\_pad\_t inpad) } \]

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:

- \( hid\_t type\_id \) 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

```fortran
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 type_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 type_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

```fortran
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:**

```c
herr_t H5Tset_offset(hid_t type_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):

<table>
<thead>
<tr>
<th>Byte Position</th>
<th>Big−Endian Offset=0</th>
<th>Big−Endian Offset=16</th>
<th>Little−Endian Offset=0</th>
<th>Little−Endian Offset=16</th>
</tr>
</thead>
<tbody>
<tr>
<td>0:</td>
<td>[ pad]</td>
<td>[0x11]</td>
<td>[0x22]</td>
<td>[ pad]</td>
</tr>
<tr>
<td>1:</td>
<td>[ pad]</td>
<td>[0x22]</td>
<td>[0x11]</td>
<td>[ pad]</td>
</tr>
<tr>
<td>2:</td>
<td>[0x11]</td>
<td>[ pad]</td>
<td>[ pad]</td>
<td>[0x22]</td>
</tr>
<tr>
<td>3:</td>
<td>[0x22]</td>
<td>[ pad]</td>
<td>[ pad]</td>
<td>[0x11]</td>
</tr>
</tbody>
</table>

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 type_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

```fortran
SUBROUTINE h5tset_offset_f(type_id, offset, hdferr)
  IMPLICIT NONE
  INTEGER(HID_T), INTENT(IN) :: type_id ! Datatype identifier
  INTEGER, 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 type_id, H5T_order_t order)

Purpose:

    Sets the byte ordering of an atomic datatype.

Description:

H5Tset_order sets the byte ordering of an atomic datatype. Byte orderings currently supported are:

- H5T_ORDER_LE (0)  
  Little–endian byte ordering (default).
- H5T_ORDER_BE (1)   
  Big–endian byte ordering.
- H5T_ORDER_VAX (2)  
  VAX mixed byte ordering (not currently supported).

Parameters:

- hid_t type_id  
  Identifier of datatype to set.
- H5T_order_t order 
  Byte ordering 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_overflow  

**Signature:**

```c
herr_t H5Tset_overflow(H5T_overflow_t func)
```

**Purpose:**

Sets the overflow handler to a specified function.

**Description:**

H5Tset_overflow sets the overflow handler to be the function specified by `func`. `func` will be called for all datatype conversions that result in an overflow.

See the definition of `H5T_overflow_t` in `H5Tpublic.h` for documentation of arguments and return values. The prototype for `H5T_overflow_t` is as follows:

```c
herr_t (*H5T_overflow_t)(hid_t src_id, hid_t dst_id, void *src_buf, void *dst_buf);
```

The NULL pointer may be passed to remove the overflow handler.

**Parameters:**

- `H5T_overflow_t func` Overflow function.

**Returns:**

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

**Fortran90 Interface:**

None.
**Name:** H5Tset_pad  

**Signature:**

```c
herr_t H5Tset_pad(hid_t type_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 type_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`

```fortran
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 type_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 type_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, 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 type_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 type_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.

Fortran Interface: h5tset_sign_f

```fortran
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:

```c
herr_t H5Tset_size(hid_t type_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 in size, but may not shrink. All datatypes must have a positive size.

Parameters:
- `hid_t type_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

```fortran
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:

```c
herr_t H5Tset_strpad(hid_t type_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 type_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

```fortran
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:**  
```c
herr_t H5Tset_tag(hid_t type_id const char *tag )
```

**Purpose:**  
Tags an opaque datatype.

**Description:**  
H5Tset_tag tags an opaque datatype `type_id` with a descriptive ASCII identifier, `tag`.

**Parameters:**  
- `hid_t type_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: h5set_tag_f**

```fortran
SUBROUTINE h5set_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 h5set_tag_f
```
Name: H5Tunregister

Signature:

```c
herr_t H5Tunregister(H5T_conv_t func)
```

Purpose:

Removes a conversion function from all conversion paths.

Description:

H5Tunregister removes a conversion function from all conversion paths.

The conversion function pointer type declaration is described in H5Tregister.

Parameters:

- `H5T_conv_t func` Function to remove from conversion paths.

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_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:

herr_t H5Zfilter_avail(H5Z_filter_t filter)

Purpose:
Detemines 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 non-negative value 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_flags )  
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_flags 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  
        H5Z_FILTER_CONFIG_DECODE_ENABLED  
            Decoding is enabled for this filter  
    (These flags are defined in the HDF5 Library source code file H5Zpublic.h)  
A bitwise AND of the returned filter_config_flags 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_flags  
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_flags 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_flags  
        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                                    Fortran90
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_func, set_local_func, and filter_func. This call to H5Zregister, registering the filter with the library, is the second step. The can_apply_func and set_local_func fields can be set to NULL if they are not required for the filter being registered.

H5Zregister accepts a single parameter, the filter_class data structure, which is defined as follows:

```c
typedef struct H5Z_class_t {
    H5Z_filter_t filter_id;
    const char *comment;
    H5Z_can_apply_func_t can_apply_func;
    H5Z_set_local_func_t set_local_func;
    H5Z_func_t filter_func;
} H5Z_class_t;
```

filter_id is the identifier for the new filter. This is a user-defined value between H5Z_FILTER_RESERVED and H5Z_FILTER_MAX, both of which are defined in the HDF5 source file H5Zpublic.h.

comment is used for debugging, may contain a descriptive name for the filter, and may be the null pointer.

can_apply_func, 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_func, 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_func, 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.

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:

```c
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:

```c
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:

```c
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.

Note:
The H5Zregister interface is substantially revised from the HDF5 Release 1.4.x series. The H5Z_class_t struct and the set local and can apply callback functions first appeared in HDF5 Release 1.6.

Parameters:
const H5Z_class_t filter_class   IN: 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 is substantially revised in Release 1.6.0 with a new H5Z_class_t struct and new set local and can apply callback functions.
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 −− Compares two HDF5 files and reports the differences.
  ✦ h5repack −− Copies an HDF5 file to a new file with or without compression/chunking.
  ✦ h5perf −− Measures HDF5 performance
  ✦ 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

• 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://hdf.ncsa.uiuc.edu/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://hdf.ncsa.uiuc.edu/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://hdf.ncsa.uiuc.edu/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", "multi", and "stream". 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 preceded 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).

\begin{verbatim}
141 142 143 000 060 061 062 012
a b c \0 0 1 2 \\
\end{verbatim}

h5dump prints them as "abc\000012\n". But if h5dump prints NUL as \0, the output is "abc\000012\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://hdf.ncsa.uiuc.edu/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 contents of the boot block.
- **-H** or **--header**
  - Print the header only; no data is displayed.
- **-A**
  - 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-bytes integer datasets as ASCII.
- **-e**
  - 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**
  - Do not print array indices with data.
- **-p** or **--properties**
  - Print dataset filters, storage layout, and fill value.
- **-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.
- **-t** T or **--datatype=T**
  - Print the specified named datatype.
Set the number of columns of output.

Output XML using XML schema (default) instead of DDL.

Output XML using XML DTD instead of DDL.

In XML output, refer to the DTD or schema at U instead of the default schema/DTD.

In XML output, (XML Schema) use qualified names in the XML:
"": no namespace, default: "hdf5:"

The next four options enable subsetting, which is accomplished by selecting a hyperslab from a dataset. These options mirror the techniques used by an HDF5 application when performing hyperslab selection. The start and count parameters are mandatory if subsetting is to be performed; the stride and block parameters are optional and will default to 1 (one).

Offset of start of subsetting selection. Default: the beginning of the dataset.

Hyperslab stride. Default: 1 in all dimensions.

Number of blocks to include in the selection.

Size of block in hyperslab. Default: 1 in all dimensions.

Indicates that all following arguments are non-options. E.g., to dump a file called `−f', use h5dump -- −f.

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", "multi", and "stream". 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

T — The name of the datatype

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.

Subsetting parameters can also be expressed in a convenient compact form, as follows:

All of the semicolons (;) are required, even when a parameter value is not specified. When not specified, default parameter values are used.
Examples:

1. Dumping the group /GroupFoo/GroupBar in the file quux.h5:

   ```
   h5dump -g /GroupFoo/GroupBar quux.h5
   ```

2. Dumping the dataset Fnord in the group /GroupFoo/GroupBar in the file quux.h5:

   ```
   h5dump -d /GroupFoo/GroupBar/Fnord quux.h5
   ```

3. Dumping the attribute metadata of the dataset Fnord which is in group /GroupFoo/GroupBar in the file quux.h5:

   ```
   h5dump -a /GroupFoo/GroupBar/Fnord/metadata quux.h5
   ```

4. Dumping the attribute metadata which is an attribute of the root group in the file quux.h5:

   ```
   h5dump -a /metadata quux.h5
   ```

5. Producing an XML listing of the file bobo.h5:

   ```
   h5dump --xml bobo.h5 > bobo.h5.xml
   ```

6. Dumping 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
   ```

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://hdf.ncsa.uiuc.edu/DTDs/HDF5−File.xsd
♦ HDF5 XML information at http://hdf.ncsa.uiuc.edu/HDF5/XML/
**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.

- `-wN` 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.
**Tool Name:** h5diff  
**Syntax:**
```
h5diff file1 file2 [OPTIONS] [object1 [object2 ]]
```
**Purpose:**  
Compares two HDF5 files and reports the differences.

**Description:**

h5diff is a command line tool that compares two HDF5 files, *file1* and *file2*, and reports the differences between them.

Optionally, h5diff will compare two objects within these files. If only one object, *object1*, is specified, h5diff will compare *object1 in file1* with *object1 in file2*. In two objects, *object1* and *object2*, are specified, h5diff will compare *object1 in file1* with *object2 in file2*. These objects must be HDF5 datasets.

*object1* and *object2* must be expressed as absolute paths from the respective file’s root group.

h5diff has the following four modes of output:

- **Normal mode**: print the number of differences found and where they occurred.
- **Report mode** (−r): print the above plus the differences.
- **Verbose mode** (−v): print the above plus a list of objects and warnings.
- **Quiet mode** (−q): do not print output (h5diff always returns an exit code of 1 when differences are found).

Additional information, with several sample cases, can be found in the document *H5diff Examples*.

**Options and Parameters:**

- **file1**  
  The HDF5 files to be compared.

- **file2**  

- **−h**  
  help message.

- **−r**  
  Report mode. Print the differences.

- **−v**  
  Verbose mode. Print the differences, list of objects, warnings.

- **−q**  
  Quiet mode. Do not print output.

- **−n count**  
  Print difference up to count differences, then stop. count must be a positive integer.

- **−d 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**  
  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 \(1 - \frac{b}{a}\) is greater than relative (e.g., \(|1 - \frac{b}{a}| > \text{relative}\), where *a* is a value in *file1* and *b* is a value in *file2*).

- **object1**  

- **object2**  
  Specific object(s) within the files to be compared.
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
file1 and file2 can be the same file. Use:
   h5diff file1 file1 /g1/dset1 /g1/dset2
to compare /g1/dset1 and /g1/dset2 in the same file

History:
Release Command Line Tool
1.6.0 Tool introduced in this release.
**Tool Name:** h5repack

**Syntax:**

```
h5repack -i file1 -o file2 [−h] [−v] [−f 'filter'] [−l 'layout'] [−m number][−e file]
```

**Purpose:**
Copies an HDF5 file to a new file with or without compression/chunking.

**Description:**

h5repack is a command line tool that applies HDF5 filters to an input file `file1`, saving the output in a new file, `file2`.

'$filter'$ is a string with the format

```
<list of objects> : <name of filter> = <filter parameters>.
```

$<list of objects>$ is a comma separated list of object names meaning apply compression only to those objects. If no object names are specified, the filter is applied to all objects

$<name of filter>$ can be:

- **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
- **NONE**, to remove the filter

$<filter parameters>$ is optional compression info

- **SHUF** (no parameter)
- **FLET** (no parameter)
- **GZIP**=<deflation level> from 1–9
- **SZIP**=<pixels per block,coding> (pixels per block is a even number in 2–32 and coding method is 'EC' or 'NN')

'$layout'$ is a string with the format

```
<list of objects> : <layout type>
```

$<list of objects>$ 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:

- **CHUNK**, to apply chunking layout
- **COMPA**, to apply compact layout
- **CONTI**, to apply continuous layout

$<layout parameters>$ is present for the chunk case only it is the chunk size of each dimension: `<dim_1 x dim_2 x ... dim_n>`

**Options and Parameters:**

- **file1**
  - The input HDF5 file
- **file2**
  - The output HDF5 file
- **−h**
  - help message.
- **−f filter**
  - Filter type
- **−l layout**
  - Layout type
Verbose mode. Print output (list of objects in the file, filters and layout applied).

File with the \(-f\) and \(-l\) options (only filter and layout flags)

Print only differences that are greater than the limit \(\text{delta}\). \(\text{delta}\) must be a positive number. The comparison criterion is whether the absolute value of the difference of two corresponding values is greater than \(\text{delta}\) (e.g., \(|ab| > \text{delta}\), where \(a\) is a value in file1 and \(b\) is a value in file2).

Do not apply the filter to objects which size in bytes is smaller than \(\text{number}\). If no size is specified a minimum of 1024 bytes is assumed.

**Examples:**

1) h5repack \(-i\) file1 \(-o\) file2 \(-f\) GZIP=1 \(-v\)  
   Applies GZIP compression to all objects in file1 and saves the output in file2

2) h5repack \(-i\) file1 \(-o\) file2 \(-f\) dset1:SZIP=8,NN \(-v\)  
   Applies SZIP compression only to object 'dset1'

3) h5repack \(-i\) file1 \(-o\) file2 \(-l\) dset1,dset2:CHUNK=20x10 \(-v\)  
   Applies chunked layout to objects 'dset1' and 'dset2'
Tool Name: h5repart

Syntax:
```
  h5repart [-v] [-V] [-{b|m}N{g|m|k}] source_file dest_file
```

Purpose:
Repartitions a file or family of files.

Description:
h5repart splits a single file into a family of files, 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.

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.
- `-bN`
  The I/O block size, defaults to 1kB
- `-mN`
  The destination member size or 1GB
- `source_file`
  The name of the source file
- `dest_file`
  The name of the destination files
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 must be expressed in the fixed floating form (e.g., 323.56) h5import is designed to accept scientific notation (e.g., 3.23E+02) in an ASCII, but that is not implemented in HDF5 release 1.6.

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 `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. |
| 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 ...] -o outfile
    or
    h5import infile -c config_file [infile ...] -o 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 ...] -o 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.

<table>
<thead>
<tr>
<th>Keyword Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>RANK</strong></td>
<td>The number of dimensions in the dataset. (Required)</td>
</tr>
<tr>
<td><code>rank</code></td>
<td>An integer specifying the number of dimensions in the dataset. Example: 4 for a 4-dimensional dataset.</td>
</tr>
<tr>
<td><strong>DIMENSION-SIZES</strong></td>
<td>Sizes of the dataset dimensions. (Required)</td>
</tr>
<tr>
<td><code>dim_sizes</code></td>
<td>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.</td>
</tr>
</tbody>
</table>
**PATH**  
*path*  
Path of the output dataset.  
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 not 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 *dataset1* for the first input dataset, *dataset2* for the second input dataset, *dataset3* 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 fixed notation (e.g., 325.34) in an ASCII file. |
| TEXTFPE   | Input is floating point data in scientific notation (e.g., 3.2534E+02) in an ASCII file. |
| (Not implemented in this release.) |
| 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.  
<p>| 8         | For signed and unsigned integer data: TEXTIN, TEXTUIN, IN, or UIN. (Default: 32) |
| 16        | |
| 32        | |
| 64        | |
| 32        | For floating point data: TEXTFP, TEXTFPE, or FP. (Default: 32) |
| 64        | |</p>
<table>
<thead>
<tr>
<th><strong>OUTPUT-CLASS</strong></th>
<th>A string denoting the type of output data.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>IN</strong></td>
<td>Output is signed integer data. (Default if INPUT-CLASS is IN or TEXTIN)</td>
</tr>
<tr>
<td><strong>UIN</strong></td>
<td>Output is unsigned integer data. (Default if INPUT-CLASS is UIN or TEXTUIN)</td>
</tr>
<tr>
<td><strong>FP</strong></td>
<td>Output is floating point data. (Default if INPUT-CLASS is not specified or is FP, TEXTFP, or TEXTFPE)</td>
</tr>
<tr>
<td><strong>STR</strong></td>
<td>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.)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>OUTPUT-SIZE</strong></th>
<th>An integer denoting the size of the output data, in bits.</th>
</tr>
</thead>
<tbody>
<tr>
<td>8</td>
<td>For signed and unsigned integer data: IN or UIN. (Default: Same as INPUT-SIZE, else 32)</td>
</tr>
<tr>
<td>16</td>
<td></td>
</tr>
<tr>
<td>32</td>
<td>For floating point data: FP. (Default: Same as INPUT-SIZE, else 32)</td>
</tr>
<tr>
<td>64</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>OUTPUT-ARCHITECTURE</strong></th>
<th>A string denoting the type of output architecture.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>NATIVE</strong></td>
<td>See the &quot;Predefined Atomic Types&quot; section in the &quot;HDF5 Datatypes&quot; chapter of the HDF5 User's Guide for a discussion of these architectures.</td>
</tr>
<tr>
<td><strong>STD</strong></td>
<td>Values marked with an asterisk (*) are not implemented in this release.</td>
</tr>
<tr>
<td><strong>IEEE</strong></td>
<td>(Default: NATIVE)</td>
</tr>
<tr>
<td><strong>INTEL</strong></td>
<td></td>
</tr>
<tr>
<td><strong>CRAY</strong></td>
<td></td>
</tr>
<tr>
<td><strong>MIPS</strong></td>
<td></td>
</tr>
<tr>
<td><strong>ALPHA</strong></td>
<td></td>
</tr>
<tr>
<td><strong>UNIX</strong></td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>OUTPUT-BYTE-ORDER</strong></th>
<th>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.</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>BE</strong></td>
<td>Big-endian. (Default)</td>
</tr>
<tr>
<td><strong>LE</strong></td>
<td>Little-endian.</td>
</tr>
</tbody>
</table>
The following options are disabled by default, making the default storage properties no chunking, no compression, no external storage, and no extensible dimensions.

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CHUNKED-DIMENSION-SIZES</td>
<td>Dimension sizes of the chunk for chunked output data. 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.</td>
</tr>
<tr>
<td>chunk_dims</td>
<td></td>
</tr>
<tr>
<td>COMPRESSION-TYPE</td>
<td>Type of compression to be used with chunked storage. Requires that CHUNKED-DIMENSION-SIZES be specified.</td>
</tr>
<tr>
<td>GZIP</td>
<td>Gzip compression. Other compression algorithms are not implemented in this release of h5import.</td>
</tr>
<tr>
<td>COMPRESSION-PARAM</td>
<td>Compression level. Required if COMPRESSION-TYPE is specified. 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.)</td>
</tr>
<tr>
<td>1 through 9</td>
<td></td>
</tr>
<tr>
<td>EXTERNAL-STORAGE</td>
<td>Name of an external file in which to create the output dataset. Cannot be used with CHUNKED-DIMENSIONS-SIZES, COMPRESSION-TYPE, OR MAXIMUM-DIMENSIONS.</td>
</tr>
<tr>
<td>external_file</td>
<td>A string specifying the name of an external file.</td>
</tr>
<tr>
<td>MAXIMUM-DIMENSIONS</td>
<td>Maximum sizes of all dimensions. Requires that CHUNKED-DIMENSION-SIZES be specified.</td>
</tr>
<tr>
<td>max_dims</td>
<td>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.</td>
</tr>
</tbody>
</table>
The help option:
The help option, expressed as one of

```
h5import -h
```
or
```
h5import -help
```
prints the h5import usage summary

```
h5import -h[elp], OR
h5import <infile> <options> [<infile> <options>...]  
-o[utfile] <outfile>
```
then exits.

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 not 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
```
```
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:**

```bash
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

```bash
−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:

```bash
h4toh5
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. (If this mapping changes between HDF5 Library releases, a more up-to-date version may be available at Mapping HDF4 Objects to HDF5 Objects on the HDF FTP server.)

In this initial version, h4toh5 converts the following HDF4 objects:

<table>
<thead>
<tr>
<th>HDF4 Object</th>
<th>Resulting HDF5 Object</th>
</tr>
</thead>
<tbody>
<tr>
<td>SDS</td>
<td>Dataset</td>
</tr>
<tr>
<td>GR, RI8, and RI24 image</td>
<td>Dataset</td>
</tr>
<tr>
<td>Vdata</td>
<td>Dataset</td>
</tr>
<tr>
<td>Vgroup</td>
<td>Group</td>
</tr>
<tr>
<td>Annotation</td>
<td>Attribute</td>
</tr>
<tr>
<td>Palette</td>
<td>Dataset</td>
</tr>
</tbody>
</table>

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:** h5perf  

**Syntax:**  

```
h5perf [-h | --help]
h5perf [options]
```

**Purpose:**  
Tests Parallel HDF5 performance. 

**Description:**  

h5perf provides tools for testing the performance of the Parallel HDF5 library. 

The following environment variables have the following effects on h5perf behavior: 

- **HDF5_NOCLEANUP**  
  If set, h5perf does not remove data files. (Default: Remove) 

- **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**  
  Specifies the block size within the transfer buffer. (Default: 128K) 
  **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 block size is set by the \(-x\) (or \(--\text{min-xfer-size}\)) and \(-X\) (or \(--\text{max-xfer-size}\)) options.

The pattern in which the blocks are written to the file is described in the discussion of the \(-I\) (or \(--\text{interleaved}\)) option.

\(-c\), \(--\text{chunk}\)

Creates HDF5 datasets in chunked layout. (Default: Off)

\(-C\), \(--\text{collective}\)

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 in each transfer buffer will be written at once with an MPI derived type.

\(-d\ N\), \(--\text{num-dsets}=N\)

Sets the number of datasets per file. (Default: 1)

\(-D\ \text{debug_flags}\), \(--\text{debug}=\text{debug_flags}\)

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: \(--\text{debug}=2,r,t\) specifies to run a moderate level of debugging while collecting raw data I/O throughput information and verifying the correctness of the data.

\(-e\ \text{size}\), \(--\text{num-bytes}=\text{size}\)

Specifies the number of bytes per process per dataset. (Default: 256K)

\(-F\ N\), \(--\text{num-files}=N\)

Specifies the number of files. (Default: 1)

\(-i\ N\), \(--\text{num-iterations}=N\)

Sets the number of iterations to perform. (Default: 1)

\(-I\), \(--\text{interleaved}\)

Sets interleaved block I/O. (Default: Contiguous block I/O)

Interleaved vs. Contiguous blocks in a parallel environment:

When contiguous blocks are written to a dataset, the dataset is divided into \(m\) regions, where \(m\) is the number of processes writing separate portions of the dataset. Each process then writes data to its own 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 has been allocated for the first block of each process. Space is then allocated for the second block of the first process, the second block of the second process, etc.
For example, in the case of a 4 process run with 1M bytes–per–process, 256K transfer buffer size, and 64KB block size, 16 contiguous blocks per process would be written to the file in the manner

```
1111111111111122222222222333333333333334444444444444444
```

while 16 interleaved blocks per process would be written to the file as

```
1234123412341234123412341234123412341234123412341234123412341234
```

If collective I/O is turned on, all of the four blocks per transfer buffer will be written in one collective I/O call.

```
-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: 128K)

-X size, --max-xfer-size=size
Sets the maximum transfer buffer size. (Default: 1M)

History:

Release Command Line Tool
1.6.0 Tool introduced in this release.
**Tool Name:** h5jam/h5unjam

**Syntax:**

```plaintext
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:**

Add user block to front of an HDF5 file, to create a new concatenated file.
Split 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 stdout and the HDF5 file to an HDF5 file with a header at byte 0 (i.e., with no user block).

If `out_file.h5` is given, a new file is created with the `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.

**Example Usage**

Create new file, `newfile.h5`, with the text in file `mytext.txt` as the user block for the HDF5 file `file.h5`.

```plaintext
h5jam -u mytext.txt -i file.h5 -o newfile.h5
```

Add text in file `mytext.txt` to front of HDF5 dataset, `file.h5`.

```plaintext
h5jam -u mytext.txt -i file.h5
```

Overwrite the user block (if any) in `file.h5` with the contents of `mytext.txt`.

```plaintext
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 file to `wo_ub.h5`.

```
    h5unjam -i with_ub.h5 -u user_block.txt -i 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**

This tool copies 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 file through byte copies, i.e., it does not read or interpret the HDF5 objects.
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 Names:** h5cc and h5pcc  
**Syntax:**  
```
h5cc [OPTIONS] compile_line  
h5pcc [OPTIONS] compile_line
```
  
**Purpose:**  
Helper scripts to compile HDF5 C 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`  
  *Print 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 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 Names: 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 progl.f90 and prog2.f90 and uses the HDF5 Fortran library, would be as follows. h5pfc is used in an identical manner.

# h5fc −c progl.f90
# h5fc −c prog2.f90
# h5fc −o hdf_prog progl.o prog2.o

Options and Parameters:

−help
Print 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 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 mpiCC by 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++ subsumes 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. Use the -show option to see the details.

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_I8LE  
  H5T_STD_I16BE  
  H5T_STD_I16LE  
  H5T_STD_I32BE  
  H5T_STD_I32LE  
  H5T_STD_I64BE  
  H5T_STD_I64LE  
  H5T_STD_U8BE  
  H5T_STD_U8LE  
  H5T_STD_U16BE  
  H5T_STD_U16LE  
  H5T_STD_U32BE  
  H5T_STD_U32LE  
  H5T_STD_U64BE  
  H5T_STD_U64LE  
  H5T_STD_B8BE  
  H5T_STD_B8LE  
  H5T_STD_B16BE  
  H5T_STD_B16LE  
  H5T_STD_B32BE  
  H5T_STD_B32LE  
  H5T_STD_B64BE  
  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_I16  H5T_INTEL_I32  H5T_INTEL_I64
H5T_INTEL_U8  H5T_INTEL_U16  H5T_INTEL_U32  H5T_INTEL_U64
H5T_INTEL_B8  H5T_INTEL_B16  H5T_INTEL_B32  H5T_INTEL_B64
H5T_INTEL_F32  H5T_INTEL_F64

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_I16  H5T_ALPHA_I32  H5T_ALPHA_I64
H5T_ALPHA_U8  H5T_ALPHA_U16  H5T_ALPHA_U32  H5T_ALPHA_U64
H5T_ALPHA_B8  H5T_ALPHA_B16  H5T_ALPHA_B32  H5T_ALPHA_B64
H5T_ALPHA_F32  H5T_ALPHA_F64

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_I16  H5T_MIPS_I32  H5T_MIPS_I64
H5T_MIPS_U8  H5T_MIPS_U16  H5T_MIPS_U32  H5T_MIPS_U64
H5T_MIPS_B8  H5T_MIPS_B16  H5T_MIPS_B32  H5T_MIPS_B64
H5T_MIPS_F32  H5T_MIPS_F64
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_SHORT
H5T_NATIVE_USHORT
H5T_NATIVE_INT
H5T_NATIVE_UINT
H5T_NATIVE_LONG
H5T_NATIVE_ULONG
H5T_NATIVE_LLONG
H5T_NATIVE_ULLONG
H5T_NATIVE_FLOAT
H5T_NATIVE_DOUBLE
H5T_NATIVE_LDOUBLE
H5T_NATIVE_B8
H5T_NATIVE_B16
H5T_NATIVE_B32
H5T_NATIVE_B64
H5T_NATIVE_INT8
H5T_NATIVE_UINT8
H5T_NATIVE_INT_LEAST8
H5T_NATIVE_UINT_LEAST8
H5T_NATIVE_INT_FAST8
H5T_NATIVE_UINT_FAST8
H5T_NATIVE_INT32
H5T_NATIVE_UINT32
H5T_NATIVE_INT_LEAST32
H5T_NATIVE_UINT_LEAST32
H5T_NATIVE_INT_FAST32
H5T_NATIVE_UINT_FAST32
H5T_NATIVE_INT64
H5T_NATIVE_UINT64
H5T_NATIVE_INT_LEAST64
H5T_NATIVE_UINT_LEAST64
H5T_NATIVE_INT_FAST64
H5T_NATIVE_UINT_FAST64

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

\[
\begin{align*}
\text{H5T_NATIVE_INTEGER} & \\
\text{H5T_NATIVE_REAL} & \\
\text{H5T_NATIVE_DOUBLE} & \\
\text{H5T_NATIVE_CHARACTER} & \\
\end{align*}
\]

• 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

\[
\begin{align*}
\text{H5T_STD_I8BE} & \quad \text{H5T_STD_U8BE} & \quad \text{H5T_IEEE_F32BE} \\
\text{H5T_STD_I8LE} & \quad \text{H5T_STD_U8LE} & \quad \text{H5T_IEEE_F32LE} \\
\text{H5T_STD_I16BE} & \quad \text{H5T_STD_U16BE} & \quad \text{H5T_IEEE_F64BE} \\
\text{H5T_STD_I16LE} & \quad \text{H5T_STD_U16LE} & \quad \text{H5T_IEEE_F64LE} \\
\text{H5T_STD_I32BE} & \quad \text{H5T_STD_U32BE} & \\
\text{H5T_STD_I32LE} & \quad \text{H5T_STD_U32LE} & \\
\text{H5T_STD_I64BE} & \quad \text{H5T_STD_U64BE} & \\
\text{H5T_STD_I64LE} & \quad \text{H5T_STD_U64LE} & \\
\end{align*}
\]

• Object reference or dataset region reference

\[
\begin{align*}
\text{H5T_STD_REF_OBJ} & \\
\text{H5T_STD_REF_DSETREG} & \\
\end{align*}
\]
### HDF5 Glossary

<table>
<thead>
<tr>
<th>Term</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>atomic datatype</strong></td>
<td>A datatype which cannot be decomposed into smaller units at the API level.</td>
</tr>
<tr>
<td><strong>attribute</strong></td>
<td>A small dataset that can be used to describe the nature and/or the intended usage of the object it is attached to.</td>
</tr>
<tr>
<td><strong>chunked layout</strong></td>
<td>The storage layout of a chunked dataset.</td>
</tr>
<tr>
<td><strong>chunking</strong></td>
<td>A storage layout where a dataset is partitioned into fixed–size multi–dimensional chunks. Chunking tends to improve performance and facilitates dataset extensibility.</td>
</tr>
<tr>
<td><strong>compound datatype</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>contiguous layout</strong></td>
<td>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.</td>
</tr>
<tr>
<td><strong>data transfer property list</strong></td>
<td>The data transfer property list is used to control various aspects of the I/O, such as caching hints or collective I/O information.</td>
</tr>
<tr>
<td><strong>dataset</strong></td>
<td>A multi–dimensional array of data elements, together with supporting metadata.</td>
</tr>
</tbody>
</table>

---

**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, group, 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.