

Happendable

Happendable

intn Happendable(int32 *h_id*)

h_id IN: Access identifier returned by **Hstartwrite**

Purpose Specifies that the specified element can be appended to

Return value Returns `SUCCEED` (or 0) if data element can be appended and `FAIL` (or -1) otherwise.

Description If a data element is at the end of a file **Happendable** allows **Hwrite** to append data to it, converting it to linked-block element only when necessary.

Hbitappendable

intn Hbitappendable(int32 *h_id*)

h_id IN: Bit-access element identifier returned by **Hstartwrite**

Purpose Specifies that the given bit-access element can be appended to.

Return value Returns `SUCCEED` (or 0) if the element can be appended and `FAIL` (or -1) otherwise

Description If a dataset is at the end of a file, this routine allows **Hbitwrite** to write past the end of a file. This allows for the existence of expanding datasets without the use of linked-blocks.

Hbitread

Hbitread

intn Hbitread(int32 *h_id*, intn *count*, uint32 **data*)

<i>h_id</i>	IN:	Bit-access element identifier returned by Hstartwrite
<i>count</i>	IN:	Number of bits to bewritten
<i>data</i>	IN/OUT:	Pointer to the bits to be read will be in the lowest-order bits. Pointer to the bits read in will be in the highest-order bits.

Purpose	Reads the specfied number of bits from the specified bit-access element.
----------------	--

Return value	Returns the number of bits read if successful and <code>FAIL</code> (or <code>-1</code>) otherwise
---------------------	---

Description	Hbitread buffers the bits and then reads them when appropriate through a call to Hread .
--------------------	--

Hbitseek

intn Hbitseek(int32 *h_id*, int32 *byte_offset*, intn *bit_offset*)

<i>h_id</i>	IN:	Bit-access element identifier returned by Hstartwrite
<i>byte_offset</i>	IN:	Byte offset to seek to within the bit-access element
<i>bit_offset</i>	IN:	Bit offset to seek to within the bit-access element.

Purpose	Seeks to the specified bit position within the specified bit-access element.
Return value	Returns <code>SUCCEED</code> (or 0) if successful and <code>FAIL</code> (or -1) otherwise
Description	If seeking to the fifteenth bit in a bit-element, the syntax of the call would be:

```
Hbitseek(bitid, 1 , 7);
```

If converting from a direct bit-offset variable the syntax of the call would be:

```
Hbitseek(bitid, bit_offset/8, bit_offset%8);
```

Hbitwrite

Hbitwrite

intn Hbitwrite(int32 *h_id*, intn *count*, uint32 *data*)

<i>h_id</i>	IN:	Bit-access element identifier returned by Hstartwrite
<i>count</i>	IN:	Number of bits to be written
<i>data</i>	IN:	Bits to be written (must be in the lowest-order bits)

Purpose Writes the specified number of bits to the specified bit-access element.

Return value Returns the number of bits written if successful and `FAIL` (or `-1`) otherwise

Description **Hbitwrite** buffers the bits and then reads them when appropriate through a call to **Hwrite**.

Hcache

intn Hcache(int32 *file_id*, intn *cache_switch*)

file_id IN: File identifier returned by **Hopen**
cache_switch IN: Flag to enable or disable caching

Purpose Enables low-level caching for the specified file.

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

Description If *file_id* is set to CACHE_ALL_FILES, then the value of *cache_switch* is used to modify the default file cache setting.

Valid values for *cache_switch* are: TRUE (or 1) to enable caching and FALSE (or 0) to disable caching.

Hclose/hclose

Hclose/hclose

intn Hclose(int32 *file_id*)

file_id IN: File identifier returned by **Hopen**

Purpose Closes the access path to the file.

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

Description The file identifier *file_id* is validated before the file is closed. If the identifier is valid, the function closes the access path to the file.

If there are still access identifiers attached to the file, the error `DFE_OPENAID` is placed on the error stack, `FAIL` (or -1) is returned, and the file remains open. This is a common error when developing new interfaces. Refer to the Reference Manual page on **Hendaccess** for a discussion of this problem.

FORTTRAN integer function hclose(file_id)

 integer file_id

Hdeldd

intn Hdeldd(int32 *file_id*, uint16 *tag*, uint16 *ref*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag of data descriptor to be deleted
<i>ref</i>	IN: Reference number of data descriptor to be deleted

Purpose Deletes a tag and reference number from the data descriptor list.

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

Description Once the data descriptor is removed, the data in the data object becomes inaccessible and is marked as such. To remove inaccessible data from an HDF file, use the utility `hdfpack`

Hdeldd only deletes the specified tag and reference number from the data descriptor list. Data objects containing the deleted tag and reference number are not automatically updated. For example, if the tag and reference number deleted from the descriptor list referenced an object in a vgroup, the tag and reference number will still exist in the vgroup even though the data is inaccessible.

Hdupdd

Hdupdd

intn Hdupdd(int32 *file_id*, uint16 *tag*, uint16 *ref*, uint16 *old_tag*, uint16 *old_ref*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag for the duplicate data descriptor
<i>ref</i>	IN: Reference number for the duplicate data descriptor
<i>old_tag</i>	IN: Tag for original data descriptor
<i>old_ref</i>	IN: Reference number for the original data descriptor

Purpose Duplicates the specified data descriptor in the data descriptor list.

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

Description Data descriptors contain the following information: tag, reference number, length, and offset. When a data descriptor is duplicated, its length and offset are added to the data descriptor list under a new tag and reference number.

Because duplicate data descriptors contain the same length and offset information, both descriptors point to the same data element in the file. In other words, duplicating a data descriptor does not duplicate the data contained in the original data object. It simply creates a second data object by redundantly pointing to the same data location in the file.

The tag and reference number for the duplicate data descriptor are assigned when **Hdupdd** is called. If the specified tag and reference number already exist in the file, the original data descriptor is not duplicated and **Hdupdd** returns an error.

Hendaccess

intn Hendaccess(int32 *h_id*)

h_id IN: Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**

Purpose Terminates access to a data object by disposing of the access identifier.

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

Description The number of active access identifiers is limited to `MAX_ACC` as defined in the `hlimits.h` header file. Because of this restriction, it is very important to call **Hendaccess** immediately following the last operation on a data element.

When developing new interfaces, a common mistake is to omit calling **Hendaccess** for all of the elements accessed. When this happens, **Hclose** will return `FAIL`, and a dump of the error stack will report the number of active access identifiers. Refer to the Reference Manual page on **HEprint**.

This is a difficult problem to debug because the low levels of the HDF library cannot determine who and where an access identifier was originated. As a result, there is no automated method of determining which access identifiers have yet to be released.

Hendbitaccess

Hendbitaccess

intn Hendbitaccess(int32 *h_id*, intn *flushbit*)

h_id IN: Identifier of the bit-access element to be disposed of

flushbit IN: Specifies how the leftover bits are to be flushed

Purpose Disposes of the specified bit-access file element.

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

Description If called after a bit-write operation, **Hendbitaccess** flushes all buffered bits to the dataset, then calls **Hendaccess**..

"Leftover bits" are bits that have been buffered, but are fewer than the number of bits defined by `BITNUM`, which is usually set to 8.

Valid codes for *flushbit* are:

0 - flush with zeros

1 - flush with ones

-1 - dispose of leftover bits

Hexist

intn Hexist(int32 *h_id*, uint16 *search_tag*, uint16 *search_ref*)

<i>h_id</i>	IN:	Access identifier returned by Hstartread , Hstartwrite , or Hnextread
<i>search_tag</i>	IN:	Tag of the object to be searched for
<i>search_ref</i>	IN:	Reference number of the object to be searched for

Purpose Locates an object in an HDF file.

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

Description Simple interface to **Hfind** that determines if a given tag/reference number pair exists in a file. Wildcards apply..

Hfind performs all validity checking; this is just a *very* simple wrapper around it.

Hfdinquire

Hfdinquire

intn Hfdinquire(int32 *file_id*, char ***filename*, intn **access*, intn **attach*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>filename</i>	OUT: Complete path and filename for the file
<i>access</i>	OUT: Access mode file is opened with
<i>attach</i>	OUT: Number of access identifiers attached to the file

Purpose Returns file information through a reference of its file identifier.

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

Description Gets the complete path name, access mode, and number of access identifiers associated with a file. The *filename* parameter is a pointer to a character pointer which will be modified when the function returns. Upon completion, *filename* is set to point to the file name in internal storage. All output parameters must be non-null pointers.

Hfind

```
intn Hfind(int32 file_id, uint16 search_tag, uint16 search_ref, uint16 *find_tag, uint16 *find_ref,
          int32 *find_offset, int32 *find_length, intn direction)
```

<i>file_id</i>	IN:	File identifier returned by Hopen
<i>search_tag</i>	IN:	The tag to search for or <code>DF_TAG_WILDCARD</code>
<i>search_ref</i>	IN:	Reference number to search for or <code>DF_REF_WILDCARD</code>
<i>find_tag</i>	IN/OUT:	If (* <i>find_tag</i> == 0) and (* <i>find_ref</i> == 0) then start the search from either the beginning or the end of the file. If the object is found, the tags of the object will be returned here.
<i>find_ref</i>	IN/OUT:	If (* <i>find_tag</i> == 0) and (* <i>find_ref</i> == 0) then start the search from either the beginning or the end of the file. If the object is found, the reference numbers of the object will be returned here.
<i>find_offset</i>	OUT:	Offset of the data element found
<i>find_length</i>	OUT:	Length of the data element found
<i>direction</i>	IN:	Direction to search in <code>DF_FORWARD</code> searches forward from the current location, and <code>DF_BACKWARD</code> searches backward from the current location

Purpose Locates the next object to be searched for in an HDF file.

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

Description **Hfind** searches for the next data element that matches the specified tag and reference number. Wildcards apply. If *direction* is `DF_FORWARD`, searching is forward from the current position in the file, otherwise `DF_BACKWARD` specifies backward searches from the current position in the file.

If *find_tag* and *find_ref* are both set to 0, this indicates the beginning of a search, and the search will start from the beginning of the file if the direction is `DF_FORWARD` and from the end of the file if the direction is `DF_BACKWARD`.

Hgetbit

Hgetbit

intn Hgetbit(int32 *h_id*)

h_id IN: Bit-access element identifier

Purpose Reads one bit from the specified bit-access element.

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

Description This function is a wrapper for **Hbitread**.

Hgetelement

int32 Hgetelement(int32 *file_id*, uint16 *tag*, uint16 *ref*, uint8 **data*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag of the data element to be read
<i>ref</i>	IN: Reference number of the data element to be read
<i>data</i>	OUT: Buffer the element will be read into
Purpose	Reads the data element for the specified tag and reference number and writes it to the <i>data</i> buffer.
Return value	Returns the number of bytes read if successful and <code>FAIL</code> (or <code>-1</code>) otherwise.
Description	It is assumed that the space allocated for the buffer is large enough to hold the data.

Hgetfileversion

Hgetfileversion

```
intn Hgetfileversion(int32 file_id, uint32 *major_v, uint32 *minor_v, uint32 *release, char
                        string[])
```

file_id IN: File identifier returned by **Hopen**

major_v OUT: Major version number

minor_v OUT: Minor version number

release OUT: Release number

string OUT: Version number text string

Purpose Retrieves version information for an HDF file.

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

Description It is still an open question as to what exactly the version number of a file should mean, so we recommend that code not depend on this buffer. The *string* argument is limited to a length of `LIBVSTR_LEN` characters as defined in `hfile.h`

Hgetlibversion

intn Hgetlibversion(uint32 **major_v*, uint32 **minor_v*, uint32 **release*, char *string*[])

major_v OUT: Major version number

minor_v OUT: Minor version number

release OUT: Release number

string OUT: Version number text string

Purpose Retrieves the version information of the current HDF library.

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

Description The version information is compiled into the HDF library, so it is not necessary to have any open files for this function to execute. The *string* buffer is limited to a length of `LIBVSTR_LEN` characters as defined in `hfile.h`

Hinquire

intn Hinquire(int32 *h_id*, int32 **file_id*, uint16 **tag*, uint16 **ref*, int32 **length*, int32 **offset*, int32 **position*, int16 **access*, int16 **special*)

<i>h_id</i>	IN: Access identifier returned by Hstartread , Hstartwrite , or Hnextread
<i>file_id</i>	OUT: File identifier returned by Hopen
<i>tag</i>	OUT: Tag of the element pointed to
<i>ref</i>	OUT: Reference number of the element pointed to
<i>length</i>	OUT: Length of the element pointed to
<i>offset</i>	OUT: Offset of the element in the file
<i>position</i>	OUT: Current position within the data element
<i>access</i>	OUT: The access type for this data element
<i>special</i>	OUT: Special code

Purpose Returns access information about a data element.

Return value Returns `SUCCESS` (or 0) if the access identifier points to a valid data element and `FAIL` (or -1) otherwise.

Description If *h_id* is a valid access identifier the access type (read or write) is set regardless of whether or not the return value is `FAIL` (or -1). If *h_id* is invalid, the function returns `FAIL` (or -1) and the access type is set to zero. To avoid excess information, pass `NULL` for any unnecessary pointer.

Hishdf

intn Hishdf(const char **filename*)

filename IN: Complete path and filename of the file to be checked

Purpose Determines if a file is an HDF file.

Return value Returns `TRUE` (or `1`) if the file is an HDF file and `FALSE` (or `0`) otherwise.

Description The first four bytes of a file identify it as an HDF file. It is possible that **Hishdf** will identify a file as an HDF file but **Hopen** will be unable to open the file; for example, if the data descriptor list is corrupt.

Hlength

Hlength

int32 Hlength(int32 *file_id*, uint16 *tag*, uint16 *ref*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag of the data element
<i>ref</i>	IN: Reference number of the data element

Purpose Returns the length of a data object specified by the tag and reference number.

Return value Returns the length of data element if found and `FAIL` (or `-1`) otherwise.

Description **Hlength** calls **Hstartread**, **HQuerylength**, and **Hendaccess** to determine the length of a data element. **Hlength** uses **Hstartread** to obtain an access identifier for the specified data object.

Hlength will return the correct data length for linked-block elements, however it is important to remember that the data in linked-block elements is not stored contiguously.

Hmpget

int Hmpget(int *pagesize*, int *maxcache*, int *flags*)

- pagesize* OUT: Page size to be used
- maxcache* OUT: Maximum number of pages to be cached
- flags* OUT: Optional routine flags

- Purpose** Returns the current settings of the page size and the maximum number of pages to be cached during the next file open or create operation.
- Return value** Returns SUCCEED (or 0) if successful and FAIL (or -1) otherwise.

Hmpset

int Hmpset(int *pagesize*, int *maxcache*, int *flags*)

<i>pagesize</i>	IN: Page size to be used
<i>maxcache</i>	IN: Maximum number of pages to be cached
<i>flags</i>	IN: Optional routine flags

Purpose Specifies the page size and the maximum number of pages to be cached during the next file open or create operation.

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

Description It is recommended that the value a *pagesize* be a power of 2.

The values set by **Hmpset** affect only the next file open or creation operation and does not change the file's paging settings after it has been created.

Currently the only valid values of *flag* is `MP_PAGEALL` which specifies that the whole file is to be cached, or 0 which specifies that no caching will be performed.

The *maxcache* parameter must have a value greater than 1, unless page buffering is to be deactivated. In this case use a *maxcache* value of 0.

Hnewref

uint16 Hnewref(int32 *file_id*)

file_id IN: File identifier returned by **Hopen**

Purpose Returns a reference number that can be used with any tag to produce a unique tag /reference number pair.

Return value Returns the reference number if successful and 0 otherwise.

Description Successive calls to **Hnewref** will generate reference number values that increase by one each time until the highest possible reference number has been returned. At this point, additional calls to **Hnewref** will return an increasing sequence of unused reference number values starting from 1.

Hnextread

Hnextread

intn Hnextread(int32 *h_id*, uint16 *tag*, uint16 *ref*, int *origin*)

<i>h_id</i>	IN: Access identifier returned by Hstartread or previous Hnextread
<i>tag</i>	IN: Tag to search for
<i>ref</i>	IN: Reference number to search for
<i>origin</i>	IN: Position to begin search: <code>DF_START</code> or <code>DF_CURRENT</code>

Purpose Searches for the next data descriptor that matches the specified tag and reference number.

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

Description Wildcards apply. If origin is `DF_START`, the search will start at the beginning of the data descriptor list. If origin is `DF_CURRENT`, the search will begin at the current position. Searching backwards from the end of a data descriptor list is not yet implemented.

If the search is successful, the access identifier reflects the new data element, otherwise it is not modified.

Hnumber/hnumber

int32 Hnumber(int32 file_id, uint16 tag)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag to be counted

Purpose	Returns the number of instances of a tag in a file.
Return value	Returns the number of instances of a tag in a file if successful, and <code>FAIL</code> (or <code>-1</code>) otherwise.
Description	Hnumber determines how many objects with the specified tag are in a file. To determine the total number of objects in a file, set the <i>tag</i> argument to <code>DFTAG_WILDCARD</code> . Note that a return value of zero is not a fail condition.

FORTTRAN	<code>integer function hnumber(file_id, tag)</code> <code>integer file_id, tag</code>
----------	--

Hoffset

Hoffset

int32 Hoffset(int32 *file_id*, uint16 *tag*, uint16 *ref*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag of the data element
<i>ref</i>	IN: Reference number of the data element

Purpose Returns the offset of a data element in the file.

Return value Returns the offset of the data element if the data element exists and `FAIL` (or `-1`) otherwise.

Description **Hoffset** calls **Hstartread**, **HQueryoffset**, and **Hendaccess** to determine the length of a data element. **Hoffset** uses **Hstartread** to obtain an access identifier for the specified data object.

Hoffset will return the correct offset for a linked-block element, however it is important to remember that the data in linked-block elements is not stored contiguously. The offset returned by **Hoffset** only reflects the position of the first data block.

Hoffset should not be used to determine the offset of an external element. In this case, **Hoffset** returns zero, an invalid offset for HDF files.

Hopen/hopen

```
int32 Hopen(char *filename, intn access, int16 n_dds)
```

<i>filename</i>	IN: Complete path and filename for the file to be opened
<i>access</i>	IN: Access code definition (preceded by DFACC_)
<i>n_dds</i>	IN: Number of data descriptors in a block if a new file is to be created

Purpose Provides an access path to an HDF file by reading all the data descriptor blocks into memory.

Return value Returns the file identifier if successful and FAIL (or -1) otherwise.

Description If given a new file name, **Hopen** will create a new file using the specified access type and number of data descriptors. If given an existing file name, **Hopen** will open the file using the specified access type and ignore the *n_dds* argument.

If *n_dds* is set to 0, the number of data descriptors will be defined as the machine default.

HDF provides several access code definitions:

DFACC_READ - Open for read only. If file does not exist, error

DFACC_CREATE - If file exists, delete it, then open a new file for read/write.

DFACC_WRITE - Open for read/write. If file does not exist, create it.

If a file is opened and an attempt is made to reopen the file using DFACC_CREATE, HDF will issue the error code DFE_ALROPEN. If the file is opened with read-only access and an attempt is made to reopen the file for write access using DFACC_RDWR or DFACC_WRITE, HDF will attempt to reopen the file with read and write permissions.

Upon successful exit, the specified file is opened with the relevant permissions, the data descriptors are set up in memory, and the associated *file_id* is returned. For new files, the appropriate file headers are also set up.

FORTTRAN

```
integer function hopen(filename, access, n_dds)

character* (*) filename
integer access, n_dds
```

Hputbit

Hputbit

intn Hputbit(int32 *h_id*, intn *bit*)

h_id IN: Bit-access element identifier

bit IN: Bit to be written

Purpose Writes one bit to the specified bit-access element.

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

Description This function is a wrapper for **Hbitwrite**.

Hputelement

int32 Hputelement(int32 *file_id*, uint16 *tag*, uint16 *ref*, uint8 **data*, int32 *length*)

<i>file_id</i>	IN:	File identifier returned by Hopen
<i>tag</i>	IN:	Tag of the data element to add or replace
<i>ref</i>	IN:	Reference number of the data element to add or replace
<i>data</i>	IN:	Pointer to data buffer
<i>length</i>	IN:	Length of data to write
Purpose		Writes a data element or replaces an existing data element in a HDF file.
Return value		Returns the number of bytes written if successful and <code>FAIL</code> (or <code>-1</code>) otherwise.

Hread

Hread

int32 Hread(int32 *h_id*, int32 *length*, VOIDP *data*)

h_id IN: Access identifier returned by **Hstartread**, **Hstartwrite**, or **Hnextread**

length IN: Length of segment to be read

data OUT: Pointer to the data array to be read

Purpose Reads the next segment in a data element.

Return value Returns the length of segment actually read if successful and `FAIL` (or `-1`) otherwise.

Description **Hread** begins reading at the current file position, reads the specified number of bytes, and increments the current file position by one. Calling **Hread** with the *length* = 0 reads the entire data element. To reposition an access identifier before writing data, use **Hseek**.

If *length* is longer than the data element, the read operation is terminated at the end of the data element, and the number of read bytes is returned. Although only one access identifier is allowed per data element, it is possible to interlace reads from multiple data elements in the same file. It is assumed that data is large enough to hold the specified data length.

Hseek

intn Hseek(int32 *h_id*, int32 *offset*, intn *origin*)

<i>h_id</i>	IN:	Access identifier returned by Hstartread , Hstartwrite , or Hnextread
<i>offset</i>	IN:	Number of bytes to seek to from the origin
<i>origin</i>	IN:	Position of the offset origin

Purpose Sets the access pointer to an offset within a data element.

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

Description Sets the seek position for the next **Hread** or **Hwrite** operation by moving an access identifier to the specified position in a data element. The *origin* and the *offset* arguments determine the byte location for the access identifier. If *origin* is set to `DF_START`, the offset is added to the beginning of the data element. If *origin* is set to `DF_CURRENT`, the offset is added to the current position of the access identifier.

Valid values for *origin* are: `DF_START` (the beginning of the file) or `DF_CURRENT` (the current position in the file).

This routine fails if the access identifier if *h_id* is invalid or if the seek position is outside the range of the data element.

Hsetaccesstype

Hsetaccesstype

int32 Hsetaccesstype(int32 *h_id*, uintn *access_type*)

h_id IN: Access identifier returned by **Hstartread** or **Hnextread**

access_type IN: I/O access type

Purpose Sets the I/O access type (serial, parallel, etc.) for a data element.

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

Hsetlength

int32 Hsetlength(int32 *file_id*, int32 *length*)

- file_id* IN: File identifier returned by **Hopen**
- length* IN: Length of the new element

- Purpose** Specifies the length of a new HDF element.
- Return value** Returns `SUCCEED` (or 0) if successful and `FAIL` (or -1) otherwise.
- Description** This function can only be used when called after **Hstartaccess** on a new data element and before any data is written to that element.

Hshutdown

Hshutdown

int32 Hshutdown()

Purpose	Deallocates buffers previously allocated in other H routines.
Return value	Returns <code>SUCCEED</code> (or 0) if successful and <code>FAIL</code> (or -1) otherwise..
Description	Should only be called by the function HDFend .

Hstartaccess

int32 Hstartaccess(int32 *file_id*, uint16 *tag*, uint16 *ref*, uint32 *flags*)

<i>file_id</i>	IN:	File identifier returned by Hopen
<i>tag</i>	IN:	Tag to search for
<i>ref</i>	IN:	Reference number to search for
<i>flags</i>	IN:	Reference number to search for

Purpose	Prepares an access element for either reading or writing.
Return value	Returns the identifier of the access element if successful and <code>FAIL</code> (or <code>-1</code>) otherwise.
Description	The data descriptor list of the specified file is searched first. If the tag/reference number pair is found, it is <i>not</i> replaced and the seek position is presumably 0. If the pair doesn't exist, it is created.

Hstartbitread

Hstartbitread

int32 Hstartbitread(int32 *file_id*, uint16 *tag*, uint16 *ref*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag to be searched for
<i>ref</i>	IN: Reference number to be searched for

Purpose	Searches the data descriptor list for the bit-read data element with the specified tag and reference number.
----------------	--

Return value	Returns the identifier for the bit-read access element if successful and <code>FAIL</code> (or <code>-1</code>) otherwise.
---------------------	---

Description	All searches begin at the head of the data descriptor list.
--------------------	---

Wildcards can be used for any tag or reference number (`DFTAG_WILDCARD` or `DFREF_WILDCARD`). All access identifiers must be released through **Hendaccess** before closing the HDF file.

Hstartread

int32 Hstartread(int32 *file_id*, uint16 *tag*, uint16 *ref*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag to be searched for
<i>ref</i>	IN: Reference number to be searched for

Purpose	Searches the data descriptor list for the data element with the specified tag and reference number.
Return value	Returns the access identifier for a data element if successful and <code>FAIL</code> (or <code>-1</code>) otherwise.
Description	<p>All searches begin at the head of the data descriptor list.</p> <p>Wildcards can be used for any tag or reference number (<code>DFTAG_WILDCARD</code> or <code>DFREF_WILDCARD</code>). All access identifiers must be released through Hendaccess before closing the HDF file.</p>

Hstartbitwrite

Hstartbitwrite

int32 Hstartbitwrite(int32 *file_id*, uint16 *tag*, uint16 *ref*, int32 *length*)

file_id IN: File identifier returned by **Hopen**

tag IN: Tag to be written to

ref IN: Reference number to be written to

length IN: The length of the data element

Purpose Enables a bit-access element for a write operation.

Return value Returns the identifier of the bit-access element if successful and `FAIL` (or `-1`) otherwise.

Description This routine calls **Hstartwrite** for most initialization procedures. **Hstartbitwrite** only initializes the bit-level context.

Hstartwrite

int32 Hstartwrite(int32 *file_id*, uint16 *tag*, uint16 *ref*, int32 *length*)

<i>file_id</i>	IN: File identifier returned by Hopen
<i>tag</i>	IN: Tag to be written to
<i>ref</i>	IN: Reference number to be written to
<i>length</i>	IN: The length of the data element

Purpose	Positions a write access identifier at the beginning of a new or existing data element.
Return value	Returns the access identifier for a data object if successful and <code>FAIL</code> (or <code>-1</code>) otherwise.
Description	Hstartwrite searches the data descriptor list for the specified tag/reference number pair. If it exists, Hstartwrite positions an access identifier at the beginning of the existing data element. Data in the existing data element is not overwritten until a call is made to Hwrite . If the tag and reference number are not found, Hstartwrite allocates space for a new data object of length <i>length</i> and positions an access identifier at the head of the data element.

Hsync

Hsync

intn Hsync(int32 *file_id*)

file_id IN: File identifier returned by **Hopen**

Purpose Makes the disk image of the specified HDF file conform to its memory representation.

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

Description This routine is currently not recommended for use as the disk representation of an HDF file is always the same as its memory representation. However, future releases of the HDF library will employ buffering schemes and this routine will come into use at that time.

Htagnewref

int32 Htagnewref(int32 *file_id*, uint16 *tag*)

<i>file_id</i>	IN: Access identifier returned by Hstartread or Hnextread
<i>tag</i>	IN: Tag to be identified with the returned reference number

Purpose	Returns a reference number that is unique for the specified file that will correspond to the specified tag. Creates a new tag/reference number pair.
Return value	Returns the reference number if successful and 0 otherwise.
Description	Successive calls to Hnewref will generate a increasing sequence of reference number values until the highest possible reference number value has been returned. It will then return unused reference number values starting from 1 in increasing order.

Htell

Htell

int32 Htell(int32 *h_id*)

h_id IN: Access identifier returned by **Hstartread** or **Hnextread**

Purpose Returns the position of an access element within a data element.

Return value Returns the position of the data element and `FAIL` (or `-1`) otherwise.

Description Analogous to **ftell**.

Htrunc

int32 Htrunc(int32 *h_id*, int32 *trunc_len*)

h_id IN: Access identifier returned by **Hstartread** or **Hnextread**
trunc_len IN: Length to truncate element

Purpose Truncates the data object specified by the *h_id* to the length *trunc_len*.
Return value Returns the length of a data element if found and `FAIL` (or `-1`) otherwise.
Description **Htrunc** does not handle special elements.

Hwrite

int32 Hwrite(int32 *h_id*, int32 *length*, VOIDP *data*)

<i>h_id</i>	IN: Access identifier returned by Hstartwrite
<i>len</i>	IN: Length of segment to be written
<i>data</i>	IN: Pointer to the data to be written

Purpose Writes the next data segment to a specified data element.

Return value Returns the length of the segment actually written if successful and `FAIL` (or `-1`) otherwise.

Description **Hwrite** begins writing at the current position of the access identifier, writes the specified number of bytes, then moves the access identifier to the position immediately following the last accessed byte. Calling **Hwrite** with *length* = 0 results in an error condition. To reposition an access identifier before writing data, use **Hseek**.

If the space allocated in the data element is smaller than the length of data, the data is truncated to the length of the data element. Although only one access identifier is allowed per data element, it is possible to interlace writes to more than one data element in a file.