Package ‘rhdf5’

February 2, 2026

Type Package

Title R Interface to HDF5

Version 2.55.12

Description This package provides an interface between HDF5 and R.
HDF5's main features are the ability to store and access very large and/or
complex datasets and a wide variety of metadata on mass storage (disk)
through a completely portable file format. The rhdf5 package is thus suited
for the exchange of large and/or complex datasets between R and other
software package, and for letting R applications work on datasets that are
larger than the available RAM.

License Artistic-2.0

URL https://huber-group-embl.github.io/rhdf5/,
https://github.com/Huber-group-EMBL/rhdf5

BugReports https://github.com/Huber-group-EMBL/rhdf5/issues

LazyLoad true

VignetteBuilder knitr

Imports Rhdf5lib (>= 1.13.4), rhdf5filters (>= 1.15.5)

Depends R (>= 4.0.0), methods

Suggests bit64, BiocStyle, knitr, rmarkdown, testthat, bench, dplyr,
ggplot2, mockery, BiocParallel, curl

LinkingTo Rhdf5lib

SystemRequirements GNU make

biocViews Infrastructure, DataImport

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

git_url https://git.bioconductor.org/packages/rhdf5

git_branch devel

git_last_commit 2480ef1

git_last_commit_date 2025-12-10

Repository Bioconductor 3.23

Date/Publication 2026-02-01

Author Bernd Fischer [aut],
Mike Smith [aut] (ORCID: <https://orcid.org/0000-0002-7800-3848>,
Maintainer from 2017 to 2025),
Gregoire Pau [aut],
Martin Morgan [ctb],
Daniel van Twisk [ctb],
Hugo Gruson [cre] (ORCID: <https://orcid.org/0000-0002-4094-1476>),
German Network for Bioinformatics Infrastructure - de.NBI [fnd]

Maintainer Hugo Gruson <hugo.gruson@embl.de>

Contents

`H5Aclose` *Close an HDF5 attribute*

Description

Close an HDF5 attribute

Usage

```
H5Aclose(h5attribute)
```

Arguments

`h5attribute`	An object of class H5IdComponent representing a the attribute to be closed. Normally created by `H5Aopen()` or similar.

See Also

`H5Aopen()`

`H5Acreate` *Create an attribute for an HDF5 object*

Description

Creates an attribute, `name`, which is attached to the object specified by the identifier `h5obj`. The attribute name must be unique for the object.

Usage

```
H5Acreate(h5obj, name, dtype_id, h5space)
```

Arguments

`h5obj`	An object of class H5IdComponent representing a H5 object identifier (file, group, or dataset). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()`, `H5Dcreate()`, or `H5Dopen()` to create an object of this kind.
`name`	The name of the attribute (character).
`dtype_id`	A character name of a datatype. See `h5const("H5T")` for possible datatypes. Can also be an integer representing an HDF5 datatype. Only simple datatypes are allowed for attributes.
`h5space`	An object of class H5IdComponent representing a H5 dataspace. See `H5Dget_space()`, `H5Screate_simple()`, `H5Screate()` to create an object of this kind.

Value

An object of class H5IdComponent representing a H5 attribute identifier.

`H5Adelete` *Delete an specified attribute of an HDF5 object*

Description

Delete an specified attribute of an HDF5 object

Usage

```
H5Adelete(h5obj, name)
```

Arguments

`h5obj`	An object of class H5IdComponent representing a H5 object identifier (file, group, or dataset). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()`, `H5Dcreate()`, or `H5Dopen()` to create an object of this kind.
`name`	The name of the attribute (character).

`H5Aexists` *Check whether an specific attribute exists for an HDF5 object*

Description

Check whether an specific attribute exists for an HDF5 object

Usage

```
H5Aexists(h5obj, name)
```

Arguments

`h5obj`	An object of class H5IdComponent representing a H5 object identifier (file, group, or dataset). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()`, `H5Dcreate()`, or `H5Dopen()` to create an object of this kind.
`name`	The name of the attribute (character).

Value

A logical value indicating whether an attribute with name `name` exists for the object specified by `h5obj`.

`H5Aget_name` *Get the name of an HDF5 attribute object*

Description

Retrieves the name of the attribute specified by an HDF5 attribute object.

Usage

```
H5Aget_name(h5attribute)
```

Arguments

`h5attribute`	An object of class H5IdComponent representing an attribute. Normally created by `H5Aopen()` or similar.

Value

A character vector of length 1 containing the name of the attribute.

`H5Aget_space` *Get a copy of the attribute dataspace*

Description

Get a copy of the attribute dataspace

Usage

```
H5Aget_space(h5attribute)
```

Arguments

`h5attribute`	An object of class H5IdComponent representing an attribute. Normally created by `H5Aopen()` or similar.

Value

Returns an object of class H5IdComponent representing a H5 dataspace identifier

---

`H5Aget_type` *Get a copy of the attribute datatype*

Description

Get a copy of the attribute datatype

Usage

```
H5Aget_type(h5attribute)
```

Arguments

`h5attribute`	An object of class H5IdComponent representing an attribute. Normally created by `H5Aopen()` or similar.

`H5Aopen` *Open an attribute for an HDF5 object*

Description

Open an attribute for an HDF5 object

Usage

```
H5Aopen(h5obj, name)

H5Aopen_by_name(h5obj, objname = ".", name)

H5Aopen_by_idx(
  h5obj,
  n,
  objname = ".",
  index_type = h5default("H5_INDEX"),
  order = h5default("H5_ITER")
)
```

Arguments

`h5obj`	An object of class H5IdComponent representing a H5 object identifier (file, group, or dataset). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()`, `H5Dcreate()`, or `H5Dopen()` to create an object of this kind.
`name`	The name of the attribute (character).
`objname`	The name of the object the attribute belongs to.
`n`	Opens attribute number `n` in the given order and index. Indexing is C-style, base-0, so the first attribute is opened with `n=0`.
`index_type`	See `h5const("H5_INDEX")` for possible arguments.
`order`	See `h5const("H5_ITER")` for possible arguments.

Value

An object of class H5IdComponent representing a H5 attribute identifier.

`H5Aread` *Read data from an HDF5 attribute*

Description

Read data from an HDF5 attribute

Usage

```
H5Aread(h5attribute, buf = NULL, bit64conversion = c("int", "double", "bit64"))
```

Arguments

`h5attribute`	An object of class H5IdComponent representing an attribute. Normally created by `H5Aopen()` or similar.
`buf`	Optional buffer to store retrieved values. The buffer size has to fit the size of the memory space `h5spaceMem`. No extra memory will be allocated for the data. Default is `NULL` which means the function will return the attribute data.
`bit64conversion`	Defines how 64-bit integers are converted. (See the details section for more information on these options.)

Details

Internally, R does not support 64-bit integers. All integers in R are 32-bit integers. By setting bit64conversion='int', a coercing to 32-bit integers is enforced, with the risk of data loss, but with the insurance that numbers are represented as integers. bit64conversion='double' coerces the 64-bit integers to floating point numbers. doubles can represent integers with up to 54-bits, but they are not represented as integer values anymore. For larger numbers there is again a data loss. bit64conversion='bit64' is recommended way of coercing. It represents the 64-bit integers as objects of class 'integer64' as defined in the package 'bit64'. Make sure that you have installed 'bit64'. The datatype 'integer64' is not part of base R, but defined in an external package. This can produce unexpected behaviour when working with the data.

Value

If `buf=NULL` returns the contents of the attribute. Otherwise return 0 if attribute is read successfully.

`H5Awrite` *Write data to an HDF5 attribute*

Description

Write data to an HDF5 attribute

Usage

```
H5Awrite(h5attribute, buf)
```

Arguments

`h5attribute`	An object of class H5IdComponent representing an attribute. Normally created by `H5Aopen()` or similar.
`buf`	The data to be written.

`h5checkFilters` *Identifies the filters required to read a dataset If filters aren't available it will try to identify them and print the names to the user.*

Description

Identifies the filters required to read a dataset If filters aren't available it will try to identify them and print the names to the user.

Usage

```
h5checkFilters(h5id)
```

`h5closeAll` *Close open HDF5 handles*

Description

This functions can be used in two ways. Firstly, it can be passed one or more H5IdComponent objects and it'll will try to close all of them regardless of the whether they represent a file, group, dataset etc. This can be easier than making multiple calls to `H5Fclose()`, `H5Gclose()`, etc.

Usage

```
h5closeAll(...)
```

Arguments

`...`	One or more objects of class H5IdComponent which should be closed. If nothing is provided to the function, all open handles will be closed.

Details

Secondly, occasionally references to HDF5 files, groups, datasets etc can be created and not closed correctly. Maybe because a function stopped before getting to the close statement, or the open handle was not assigned to an R variable. If no arguments are provide this function identifies all open handles and closes them.

Value

Doesn't return anything. Called for the side-effect of closing open HDF5 handles.

Author(s)

Mike Smith

Examples

```
## create an empty file and then re-open it
h5File <- tempfile(pattern = "ex_h5closeAll.h5")
h5createFile(h5File)
H5Fopen(h5File)

## list all open identifiers
h5listIdentifier()

## close all open identifiers and verify
h5closeAll()
h5listIdentifier()
```

`h5constants` *HDF5 library constants.*

Description

Access to HDF5 constants.

Usage

```
h5const(type = "")

h5constType()

h5default(type = "")
```

Arguments

`type`	A character name of a group of constants.

Details

These functions provide a list of HDF5 constants that are defined in the R package. `h5constType` provides a list of group names and `h5const` gives the constants defined within a group. `h5default` gives the default choice for each group.

Value

A character vector with names of HDF5 constants or groups.

Author(s)

Bernd Fischer

Examples

```
h5constType()[1]
h5const(h5constType()[1])
```

`H5Dclose` *Close an open HDF5 dataset*

Description

Close an open HDF5 dataset

Usage

```
H5Dclose(h5dataset)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset

`H5Dcreate` *Create a new HDF5 dataset*

Description

Create a new HDF5 dataset

Usage

```
H5Dcreate(
  h5loc,
  name,
  dtype_id,
  h5space,
  lcpl = NULL,
  dcpl = NULL,
  dapl = NULL
)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`name`	Name of the dataset.
`dtype_id`	A character name of a datatype. See `h5const("H5T")` for possible datatypes. Can also be an integer representing an HDF5 datatype.
`h5space`	An object of class H5IdComponent representing a H5 dataspace. See `H5Dget_space()`, `H5Screate_simple()`, `H5Screate()` to create an object of this kind
`lcpl, dcpl, dapl`	An objects of class H5IdComponent representing HDF5 property lists. Specially these should respectively be: a link creation property list, a dataset creation property list, a dataset access property list

Value

An object of class `H5IdComponent` representing the opened dataset.

`H5Dget_create_plist` *Return a copy of the dataset creation property list for a dataset*

Description

Return a copy of the dataset creation property list for a dataset

Usage

```
H5Dget_create_plist(h5dataset)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset

---

`H5Dget_num_chunks` *Get the number of chunks in a dataset*

Description

Retrieves the number of chunks used by an HDF5 dataset.

Usage

```
H5Dget_num_chunks(h5dataset)
```

Arguments

`h5dataset`	An object of class H5IdComponent representing the dataset from which chunks will be counted.

Details

Note, this function only returns the number of chunks that actually have data written to them. It does not return the theoretical number of chunks in a dataset or intersection with a dataspace. For example, if an empty dataset is created and but no values have been written to it `H5Dget_num_chunks()` will return 0. This can be seen in the examples below.

The C API also provides an optional parameter to constrain the query by providing a dataspace selection. However this argument is not currently used at the C level and so is ommitted here.

Value

An integer value indicating the number of chunks present in the dataset or selected region.

Examples

```
file <- tempfile(fileext = ".h5")
fid <- H5Fcreate(file)

## Create a dataset that will be represented by 4 chunks if complete
h5createDataset(file, "data", dims = c(10, 10), chunk = c(5, 5), storage.mode = "integer")
did <- H5Dopen(fid, "data")
## Here we return 0 chunks as no values have been written
H5Dget_num_chunks(did)

## Now write data to half the dataset
h5writeDataset(obj = matrix(1:50, nrow = 10), h5loc = fid, name = "/data", index = list(1:10, 1:5))
## We now see it contains 2 chunks
H5Dget_num_chunks(did)

## Now write the complete dataset, overwriting the existing values
h5writeDataset(obj = matrix(201:300, nrow = 10), h5loc = fid, name = "/data", index = NULL)
## We now see it contains 4 chunks
H5Dget_num_chunks(did)

## Tidy up op handles
h5closeAll(did, fid)
```

`H5Dget_space` *Return a copy of the HDF5 dataspace for a dataset*

Description

Return a copy of the HDF5 dataspace for a dataset

Usage

```
H5Dget_space(h5dataset)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset

Value

Returns an object of class `H5IdComponent` representing a HDF5 dataspace identifier

`H5Dget_storage_size` *Find the amount of storage allocated for a dataset*

Description

`H5Dget_storage_size` returns the amount of storage, in bytes, allocated in an HDF5 file to hold a given dataset. This is the amount of space required on-disk, which not typically a good indicator of the amount of memory that will be required to read the complete dataset.

Usage

```
H5Dget_storage_size(h5dataset)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset

Value

Returns an integer giving the number of bytes allocated in the file to the dataset.

`H5Dget_type` *Return a copy of the HDF5 datatype for a dataset*

Description

Return a copy of the HDF5 datatype for a dataset

Usage

```
H5Dget_type(h5dataset)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset

`H5Dopen` *Open an existing HDF5 dataset*

Description

Open an existing HDF5 dataset

Usage

```
H5Dopen(h5loc, name, dapl = NULL)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group).
`name`	Name of the dataset to open.
`dapl`	An object of class H5IdComponent representing a H5 dataset access property list.

Value

An object of class `H5IdComponent` representing the opened dataset. To prevent memory leaks this must be closed with a call to `H5Dclose()` when no longer needed.

Examples

```
h5file <- tempfile(fileext = ".h5")
h5createFile(h5file)
h5createDataset(h5file, dataset = "A", dims = 10)

fid <- H5Fopen(h5file)
did <- H5Dopen(h5loc = fid, name = "A")
did

## rember to close open handles
H5Dclose(did)
H5Fclose(fid)
```

H5Dread *Read from an HDF5 dataset*

Description

`H5Dread()` reads a (partial) dataset from an HDF5 file into the R session.

Usage

```
H5Dread(
  h5dataset,
  h5spaceFile = NULL,
  h5spaceMem = NULL,
  buf = NULL,
  compoundAsDataFrame = TRUE,
  bit64conversion = c("int", "double", "bit64"),
  drop = FALSE
)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset.
`h5spaceFile`	An object of class `H5IdComponent` representing a HDF5 dataspace. See `H5Dget_space()`, `H5Screate_simple()`, `H5Screate()` to create an object of this kind.
`h5spaceMem`	An object of class `H5IdComponent` representing a HDF5 dataspace. See `H5Dget_space()`, `H5Screate_simple()`, `H5Screate()` to create an object of this kind. The dimensions of the dataset in the file and in memory. The dimensions in file and in memory are interpreted in an R-like manner. The first dimension is the fastest changing dimension. When reading the file with a C-program (e.g. HDFView) the order of dimensions will invert, because in C the fastest changing dimension is the last one.
`buf`	Buffer to hold the read data. The buffer size has to fit the size of the memory space `h5spaceMem`. No extra memory will be allocated for the data. A pointer to the same data is returned.
`compoundAsDataFrame`	Logical vector of length 1. If `TRUE`, a compound datatype will be coerced to a `data.frame`. This is not possible, if the dataset is multi-dimensional. Otherwise the compound datatype will be returned as a `list`. Nested compound data types will be returned as a nested `list`.
`bit64conversion`	Defines how 64-bit integers are converted. (See the details section for more information on these options.)
`drop`	Logical vector of length 1. If `TRUE`, the HDF5 object is read as a vector with NULL dim attributes. Default is `FALSE`.

Details

Internally, R does not support 64-bit integers. All integers in R are 32-bit integers. By setting bit64conversion='int', a coercing to 32-bit integers is enforced, with the risk of data loss, but with the insurance that numbers are represented as integers. bit64conversion='double' coerces the 64-bit integers to floating point numbers. doubles can represent integers with up to 54-bits, but they are not represented as integer values anymore. For larger numbers there is again a data loss. bit64conversion='bit64' is recommended way of coercing. It represents the 64-bit integers as objects of class 'integer64' as defined in the package 'bit64'. Make sure that you have installed 'bit64'. The datatype 'integer64' is not part of base R, but defined in an external package. This can produce unexpected behaviour when working with the data.

`H5Dset_extent` *Change the dimensions of an HDF5 dataset*

Description

Change the dimensions of an HDF5 dataset

Usage

```
H5Dset_extent(h5dataset, size)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset.
`size`	An integer vector with the new dimension of the dataset.

Details

This function can only be applied to datasets that meet the following criteria:

- A chunked dataset with unlimited dimensions
- A chunked dataset with fixed dimensions if the new dimension sizes are less than the maximum sizes set with maxdims

Value

A logical vector of length 1. Value will be `TRUE` if the operation was sucessful and `FALSE` otherwise.

Author(s)

Bernd Fischer, Mike Smith

`H5Dwrite` *Write data to dataset*

Description

Write data to dataset

Usage

```
H5Dwrite(h5dataset, buf, h5type = NULL, h5spaceMem = NULL, h5spaceFile = NULL)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset.
`buf`	The R object containing the data to be written to the dataset.
`h5type`	Datatype of the HDF5 dataset to be written. If left as `NULL` it will use the dataype of the R object supplied to `buf`.
`h5spaceMem`, `h5spaceFile`	H5IdComponent objects representing the memory and file dataspaces respectively. If these are left `NULL` dataspaces that match the size and shape of `h5dataset` will be used.

`H5D_extras` *Additional functions for finding details of dataset chunking.*

Description

Additional functions for finding details of dataset chunking.

Usage

```
H5Dchunk_dims(h5dataset)

H5Dis_chunked(h5dataset)
```

Arguments

`h5dataset`	Object of class H5IdComponent representing an open HDF5 dataset.

Details

These functions do not map directly to the HDF5 C API but follow the same style and are included as potentially useful additions.

- `H5Dis_chunked` tests whether a dataset is chunked.
- `H5Dchunk_dims` will return the dimensions of the dataset chunks.

Value

- `H5Dchunk_dims`: If the supplied dataset is chunked returns a vector, with length equal to the rank of the dataset, containing the size of the dataset dimensions. Returns `NULL` if the given dataset is not chunked.
- `H5Dis_chunked`: returns `TRUE` if a dataset is chunked and `FALSE` otherwise.

Author(s)

Mike Smith

`H5Fclose` *Close access to an HDF5 file*

Description

Close access to an HDF5 file

Usage

```
H5Fclose(h5file)
```

Arguments

`h5file`	H5IdComponent representing an HDF5 file ID. Typically created via `H5Fcreate()` or `H5Fopen()`.

`H5Fcreate` *Create an HDF5 file*

Description

Create an HDF5 file

Usage

```
H5Fcreate(
  name,
  flags = h5default("H5F_ACC"),
  fcpl = NULL,
  fapl = NULL,
  native = FALSE
)
```

Arguments

`name`	The name of the HDF5 file to create.
`flags`	See `h5const("H5F_ACC")` for possible arguments.
`fcpl`, `fapl`	Object object of class H5IdComponent. This should representing a file creation property list and a file access property list respectively. See `H5Pcreate()` or `H5Pcopy()` to create objects of this kind. Leaving as `NULL` will use the default HDF5 settings which are often sufficient.
`native`	An object of class `logical`. If `TRUE`, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`.

`H5Fflush` *Flush all buffers associated with a file to disk*

Description

Flush all buffers associated with a file to disk

Usage

```
H5Fflush(h5file, scope = h5default("H5F_SCOPE"))
```

Arguments

`h5file`	H5IdComponent representing any object associated with the file to be flushed.
`scope`	Specifies whether the scope of the flushing action is global (flushes the entire virtual file) or local (flushes only the specified file). Valid values are `H5F_SCOPE_GLOBAL` and `H5F_SCOPE_LOCAL`.

`H5Fget_filesize` *Find the size of an open HDF5 file*

Description

`H5Fget_filesize()` returns the size in bytes of the HDF5 file specified by `h5file`.

Usage

```
H5Fget_filesize(h5file)
```

Arguments

`h5file`	H5IdComponent representing an HDF5 file ID. Typically created via `H5Fcreate()` or `H5Fopen()`.

H5Fget_intent *Determine the read only or read/write status of an open file handle.*

Description

Determine the read only or read/write status of an open file handle.

Usage

```
H5Fget_intent(h5file)
```

Arguments

`h5file`	An object of class H5IdComponent representing a H5 file identifier. Typically produced by `H5Fopen()` or `H5Fcreate()`.

Details

The native `H5Fget_intent()` function can in theory also return the values `H5F_ACC_SWMR_WRITE` and `H5F_ACC_SWMR_READ`. However these require the underlying HDF5 library to be complied with support for single-writer/multiple-reader (SWMR), which Rhdf5lib currently is not. Hence only the two values detailed in the values section should be possible.

Value

Returns a character vector of length 1. This will either be `H5F_ACC_RDWR` (read / write) or `H5F_ACC_READONLY` (read only).

Examples

```
## use an example file and show its location
h5file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")
## open the file as read only and check this
fid <- H5Fopen(h5file, flags = "H5F_ACC_RDONLY")
H5Fget_intent(fid)
H5Fclose(fid)

## open file as read write and confirm
fid <- H5Fopen(h5file, flags = "H5F_ACC_RDWR")
H5Fget_intent(fid)
H5Fclose(fid)
```

H5Fget_name *Retrieve the name of the file to which an object belongs*

Description

Retrieve the name of the file to which an object belongs

Usage

```
H5Fget_name(h5obj)
```

Arguments

h5obj	An object of class H5IdComponent. Despite this being an H5F function, it works equally well on H5 file, group, dataset and attribute datatypes.

Examples

```
## use an example file and show its location
h5file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")
h5file

## open a file handle and confirm we can identify the file it points to
fid <- H5Fopen(h5file)
H5Fget_name(fid)

## H5Fget_name() can be applied to group and dataset handles too
gid <- H5Gopen(fid, name = "/")
did <- H5Dopen(fid, name = "DS1")
H5Fget_name(gid)
H5Fget_name(did)

## tidy up
H5Dclose(did)
H5Gclose(gid)
H5Fclose(fid)
```

H5Fget_plist *Get property lists associated with an HDF5 file*

Description

Get property lists associated with an HDF5 file

Usage

```
H5Fget_create_plist(h5file)

H5Fget_access_plist(h5file)
```

Arguments

`h5file`	An object of class H5IdComponent representing a H5 file identifier. Typically produced by `H5Fopen()` or `H5Fcreate()`.

`H5Fis_hdf5` *Determine whether a file is in the HDF5 format*

Description

`H5Fis_hdf5()` determines whether a file is in the HDF5 format.

Usage

```
H5Fis_hdf5(name, showWarnings = TRUE)
```

Arguments

`name`	Character vector of length 1, giving the path to the file to be checked.
`showWarnings`	If the file doesn't exist an warning is generated. Setting this argument to `FALSE` will suppress the warning.

Value

Returns `TRUE`, if the file is an HDF5 file, or `FALSE` otherwise. In the case the file doesn't exist, `NA` is returned

---

`H5Fopen` *Open an existing HDF5 file*

Description

Open an existing HDF5 file

Usage

```
H5Fopen(name, flags = h5default("H5F_ACC_RD"), fapl = NULL, native = FALSE)
```

Arguments

`name`	The name (or path) of the HDF5 file to be opened.
`flags`	Character string defining the access mode for opening the file.
`fapl`	H5IdComponent object representing a file access property list. Leaving this argument as `NULL` will use the default HDF5 properties.
`native`	An object of class `logical`. If `TRUE`, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be opened for reading with `native = TRUE`.

Details

Possible values for the `flags` argument are `H5F_ACC_RDWR` and `H5F_ACC_RDONLY`. Note that HDF5's "Single Write Multiple Reader (SWMR) mode is not currently supported via **rhdf5**.

`H5functions` *HDF5 General Library Functions*

Description

These low level functions provide general library functions for HDF5.

Usage

```
H5open()

H5close()

H5garbage_collect()

H5get_libversion()
```

Value

- `H5open` initializes the HDF5 library.
- `H5close` flushes all data to disk, closes all open identifiers, and cleans up memory.
- `H5garbage_collect` cleans up memory.
- `H5get_libversion` returns the version number of the HDF5 C-library.

Author(s)

Bernd Fischer, Mike Smith

Examples

```
## Not run:
H5open()
H5close()
H5garbage_collect()
H5get_libversion()

## End(Not run)
```

`H5Gclose` *Close a specified group*

Description

Close a specified group

Usage

```
H5Gclose(h5group)
```

Arguments

`h5group`	An object of class H5IdComponent representing a H5 group. Typically created via `H5Gopen()` or `H5Gcreate()`.

`H5Gcreate` *Create a new HDF5 group and link it to a location in a file*

Description

`H5Gcreate` is used to a new group and link it into a file.

Usage

```
H5Gcreate(h5loc, name)
```

Arguments

`h5loc`	An object of class H5IdComponent
`name`	Name of the new group to be created.

`H5Gcreate_anon` *Create a new HDF5 group without linking it into a file*

Description

Create a new HDF5 group without linking it into a file

Usage

```
H5Gcreate_anon(h5loc)
```

Arguments

`h5loc`	An object of class H5IdComponent specifying the file in which the new group is to be created.

Value

`H5Gcreate_anon` returns an object of class H5IdComponent representing the newly created group. However at this point is is still anonymous, and must be linked into the file structure via `H5Olink()`. If this is not done, the group will be deleted from the file when it is closed.

See Also

`H5Gcreate()`, `H5Olink()`

`H5Gget_info` *Retrieve information about a group*

Description

Retrieve information about a group

Usage

```
H5Gget_info(h5loc)

H5Gget_info_by_name(h5loc, group_name)

H5Gget_info_by_idx(
  h5loc,
  n,
  group_name = ".",
  index_type = h5default("H5_INDEX"),
  order = h5default("H5_ITER")
)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 group.
`group_name`	An additional group name specifying the group for which information is sought. It is interpreted relative to `h5loc`.
`n`	Position in the index of the group for which information is retrieved.
`index_type`	See `h5const("H5_INDEX")` for possible arguments.
`order`	See `h5const("H5_ITER")` for possible arguments.

Value

A list with group information

Examples

```
h5file <- system.file("testfiles", "multiple_dtypes.h5", package = "rhdf5")
fid <- H5Fopen(h5file)
gid <- H5Gopen(fid, "/foo")
gid
H5Gget_info(gid)
H5Gclose(gid)

## the "get_info_by" functions take the H5 object that contains the
## group(s) of interest.  We can retrieve information by index or by name
H5Gget_info_by_idx(fid, 3)
H5Gget_info_by_name(fid, "/foo")

H5Fclose(fid)
```

`H5Gopen` *Open a specified group*

Description

Open a specified group

Usage

```
H5Gopen(h5loc, name)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 file or group that contains the group to be opened.
`name`	Name of the group to open.

Value

An object of class H5IdComponent representing the opened group. When access to the group is no longer needed this should be released with `H5Gclose()` to prevent resource leakage.

See Also

`H5Gclose()`

`H5IdComponent-class` *An S4 class representing an H5 object*

Description

A class representing a HDF5 identifier handle. HDF5 identifiers represent open files, groups, datasets, dataspaces, attributes, and datatypes.

Usage

```
## S4 method for signature 'H5IdComponent'
show(object)

## S4 method for signature 'H5IdComponent,character'
e1 & e2

## S4 method for signature 'H5IdComponent'
x$name

## S4 replacement method for signature 'H5IdComponent'
x$name <- value

## S4 method for signature 'H5IdComponent'
x[i, j, ..., drop = TRUE]

## S4 replacement method for signature 'H5IdComponent'
x[i, j, ...] <- value
```

Arguments

`object`	Object of class `H5IdComponent`
`e1`	An `H5IdComponent` object representing an H5 file or group.
`e2`	Character giving the path to an HDF5 group or dataset relative to `e1`.
`x`	Object of class `H5IdComponent` representing the HDF5 dataset from which to extract element(s) or in which to replace element(s).
`name`	Character giving the path to an HDF5 group or dataset relative to `x`.
`value`	Array-like R object containing value to be inserted into the HDF5 dataset.

`i, j, ...`	Indices specifying elements to extract or replace. Indices are `numeric` vectors or empty (missing) or `NULL`. Numeric values are coerced to integer as by `base::as.integer()` (and hence truncated towards zero).
`drop`	If `TRUE` the result is coerced to the lowest possible dimension (see the examples). This only works for extracting elements, not for the replacement. See `base::drop()` for further details.

Methods (by generic)

- `show(H5IdComponent)`: Print details of the object to screen.
- `e1 & e2`: Returns a group handle or dataset handle for the group or dataset `name` in the HDF5 location `h5loc`. `h5loc` can either be a file handle as returned by H5Fopen or a group handle as e.g. returned by `h5f$g1` or `h5f$'/g1/g2'`.
- `$`: Reads the HDF5 object `name` in the HDF5 location `x`. `x` can either be a file handle as returned by `H5Fopen()` or a group handle as e.g. returned by `h5f$g1` or `h5f$'/g1/g2'`.
- `` `$`(H5IdComponent) <- value ``: Writes the assigned object to to the HDF5 file at location `e1`. `e1` can either be a file handle as returned by `H5Fopen()` or a group handle as e.g. returned by h5f$g1 or h5f$'/g1/g2's. The storage.mode of the assigned object has to be compatible to the datatype of the HDF5 dataset. The dimension of the assigned object have to be identical the dimensions of the HDF5 dataset. To create a new HDF5 dataset with specific properties (e.g. compression level or chunk size), please use the function `h5createDataset()` first.
- `[`: Subsetting of an HDF5 dataset. The function reads a subset of an HDF5 dataset. The given dimensions have to fit the dimensions of the HDF5 dataset.
- `` `[`(H5IdComponent) <- value ``: Subsetting of an HDF5 dataset. The function writes an R data object to a subset of an HDF5 dataset. The given dimensions have to fit the dimensions of the HDF5 dataset. The HDF5 dataset has to be created beforehand, e.g. by `h5createDataset()`.

Slots

`ID` `integer` of length 1. Contains the handle of C-type `hid_t`.

`native` An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

`H5Iget_name` *Retrieve the name of an object from a given identifier*

Description

Retrieve the name of an object from a given identifier

Usage

```
H5Iget_name(h5obj)
```

Arguments

`h5obj`	An object of class H5IdComponent. Can represent a file, group, dataset or attribute.

`H5Iget_type` *Find the type of an object*

Description

Possible types returned by the function are:

- `H5I_FILE`
- `H5I_GROUP`
- `H5I_DATATYPE`
- `H5I_DATASPACE`
- `H5I_DATASET`
- `H5I_ATTR`

Usage

```
H5Iget_type(h5identifier)
```

Arguments

`h5identifier`	Object of class H5IdComponent.

Value

Returns a character vector of length 1 containing the HDF5 type for the supplied identifier.

Examples

```
h5file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")
fid <- H5Fopen(h5file)
gid <- H5Gopen(fid, "/")

## identify the HDF5 types for these identifiers
H5Iget_type(fid)
H5Iget_type(gid)

## tidy up
H5Gclose(gid)
H5Fclose(fid)
```

H5Iis_valid *Determine whether an identifier is valid*

Description

An identifier is no longer valid after it has been closed.

Usage

```
H5Iis_valid(h5identifier)
```

Arguments

`h5identifier`	Object of class H5IdComponent.

Value

A logical of length 1. `TRUE` is the identifier is valid, `FALSE` if not.

Examples

```
h5file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")
fid <- H5Fopen(h5file)

## test whether the identifer to the opened file is valid
H5Iis_valid(fid)

## the file ID is no longer valid after it has been closed
H5Fclose(fid)
H5Iis_valid(fid)
```

H5Lcopy *Copy a link from one location to another*

Description

Copy a link from one location to another

Usage

```
H5Lcopy(h5loc, name, h5loc_dest, name_dest, lcpl = NULL, lapl = NULL)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group) where the new link is placed.
`name`	The name of the link to be copied.
`h5loc_dest`	An object of class H5IdComponent representing the destination file or group where a copied or moved link should be created.
`name_dest`	The name of the link to be created when copying or moving.
`lcpl, lapl`	Link creation and link access property lists. If left as `NULL` the HDF5 defaults will be used.

`H5Lcreate_external` *Create a link to an object in a different HDF5 file*

Description

`H5Lcreate_external()` creates a new external link. An external link is a soft link to an object in a different HDF5 file from the location of the link.

Usage

```
H5Lcreate_external(target_file_name, target_obj_name, link_loc, link_name)
```

Arguments

`target_file_name`	Name of the external HDF5 to link to
`target_obj_name`	Path to the object in the file specified by `target_file_name` to link to.
`link_loc`	H5IdComponent object giving the location where the new link should be created. Can represent an HDF5 file or group.
`link_name`	Name (path) of the new link, relative to the location of `link_loc`.

Examples

```
## The example below creates a new HDF5 file in a temporary director, and then
## links to the group "/foo" found in the file "multiple_dtypes.h5"
## distributed with the package.

h5File1 <- system.file("testfiles", "multiple_dtypes.h5", package = "rhdf5")
h5File2 <- tempfile(pattern = "H5L_2_", fileext = ".h5")
h5createFile(h5File2)

## open the new file & create a link to the group "/foo" in the original file
fid <- H5Fopen(h5File2)
H5Lcreate_external(
  target_file_name = h5File1, target_obj_name = "/foo",
  link_loc = fid, link_name = "/external_link"
)
H5Fclose(fid)

## check the new file has a group called "/external_link"
h5ls(h5File2)
```

H5Ldelete *Remove a link from a group*

Description

Remove a link from a group

Usage

```
H5Ldelete(h5loc, name)
```

Arguments

h5loc	An object of class H5IdComponent representing a H5 location identifier (file or group).
name	The name of the link to be deleted.

Examples

```
h5file <- tempfile(pattern = "_ex_H5L.h5")

# create an hdf5 file and a group
h5createFile(h5file)
h5createGroup(h5file, "/foo")

# reopen file and confirm "/foo" exists but "/baa" does not
fid <- H5Fopen(h5file)
H5Lexists(fid, "/foo")

# remove the link to "/foo" and confirm it no longer exists
H5Ldelete(fid, "/foo")
H5Lexists(fid, "/foo")

H5Fclose(fid)
```

`H5Lexists` *Confirm existence of a link*

Description

Confirm existence of a link

Usage

```
H5Lexists(h5loc, name)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group).
`name`	The name of the link to be checked

`H5Lget_info` *Find information about a link*

Description

`H5Lget_info()` identifies the type of link specified by the the `h5loc` and `name` arguments. This is more limited than the equivalent function in the standard HDF5 library.

Usage

```
H5Lget_info(h5loc, name)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group).
`name`	The name of the link to be queried.

Value

A character vector of length 1 giving the type of link. Possible values are: `H5L_TYPE_HARD`, `H5L_TYPE_SOFT`, `H5L_TYPE_EXTERNAL`, `H5L_TYPE_ERROR` `h5listObjects` *List all open HDF5 objects.*

Description

A list of all valid HDF5 identifiers. H5 objects should be closed after usage to release resources.

Usage

```
h5listIdentifier()

h5validObjects(native = FALSE)
```

Arguments

`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

Value

`h5validObjects` returns a list of H5IdComponent objects. `h5listIdentifier` prints the valid identifiers on screen and returns NULL.

Author(s)

Bernd Fischer, Mike Smith

Examples

```
h5File <- tempfile("ex_list_identifier.h5")

h5createFile(h5File)

# create groups
h5createGroup(h5File, "foo")

h5listIdentifier()
h5validObjects()
```

H5Lmove *Move a link within an HDF5 file*

Description

Move a link within an HDF5 file

Usage

```
H5Lmove(h5loc, name, h5loc_dest, name_dest, lcpl = NULL, lapl = NULL)
```

Arguments

`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group) where the new link is placed.
`name`	The name of the link to be moved.
`h5loc_dest`	H5IdComponent object representing the H5 location where the new link should be created.
`name_dest`	Name of the new link to be created
`lcpl, lapl`	Link creation and link access property lists to be associated with the new link. Leaving these arguments as `NULL` will use the HDF5 default property lists.

Examples

```
## create an HDF5 file with a single group
## that contains a dataset of 10 numbers
h5file <- tempfile(fileext = ".h5")
h5createFile(h5file)
h5createGroup(h5file, "/foo")
h5write(1:10, h5file, name = "/foo/vector1")
## check the structure is what we expect
h5ls(h5file)

## open the file, the group where the dataset currently is
## and the root group
fid <- H5Fopen(name = h5file)
gid1 <- H5Gopen(fid, "/foo")
gid2 <- H5Gopen(fid, "/")
## move the dataset to the root of the file and rename it
H5Lmove(gid1, "vector1", gid2, "vector_new")
h5closeAll()
## check the dataset has moved out of the foo group
h5ls(h5file)

## we can also provide the ID of the HDF5 file
## and use the "name" arguments to move between groups
fid <- H5Fopen(name = h5file)
H5Lmove(fid, "/vector_new", fid, "/foo/vector_newer")
```

```
H5Fclose(fid)
h5ls(h5file)
```

h5ls *List the content of an HDF5 file.*

Description

List the content of an HDF5 file.

Usage

```
h5ls(
  file,
  recursive = TRUE,
  all = FALSE,
  datasetinfo = TRUE,
  index_type = h5default("H5_INDEX"),
  order = h5default("H5_ITER"),
  s3 = FALSE,
  s3credentials = NULL,
  native = FALSE
)
```

Arguments

`file`	The filename (character) of the file in which the dataset will be located. You can also provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`recursive`	If `TRUE`, the content of the whole group hierarchy is listed. If `FALSE`, Only the content of the main group is shown. If a positive integer is provided this indicates the maximum level of the hierarchy that is shown.
`all`	If `TRUE`, a longer list of information on each entry is provided.
`datasetinfo`	If `FALSE`, datatype and dimensionality information is not provided. This can speed up the content listing for large files.
`index_type`	See `h5const("H5_INDEX")` for possible arguments.
`order`	See `h5const("H5_ITER")` for possible arguments.
`s3`	Logical value indicating whether the file argument should be treated as a URL to an Amazon S3 bucket, rather than a local file path.
`s3credentials`	A list of length three, providing the credentials for accessing files in a private Amazon S3 bucket.
`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

Value

h5ls returns a data.frame with the file content.

Author(s)

Bernd Fischer, Mike L. Smith

References

https://portal.hdfgroup.org/display/HDF5

See Also

h5dump()

Examples

```
h5File <- tempfile(pattern = "ex_dump.h5")
h5createFile(h5File)

# create groups
h5createGroup(h5File, "foo")
h5createGroup(h5File, "foo/foobaa")

# write a matrix
B <- array(seq(0.1, 2.0, by = 0.1), dim = c(5, 2, 2))
attr(B, "scale") <- "liter"
h5write(B, h5File, "foo/B")

# list content of hdf5 file
h5ls(h5File, all = TRUE)

# list content of an hdf5 file in a public S3 bucket

h5ls(file = "https://rhdf5-public.s3.eu-central-1.amazonaws.com/h5ex_t_array.h5", s3 = TRUE)
```

H5Oclose *Close an HDF5 object*

Description

Close an HDF5 object

Usage

```
H5Oclose(h5obj)
```

Arguments

`h5obj`	An object of class H5IdComponent representing an open HDF5 object.

See Also

`H5Oopen()`

`H5Ocopy` *Copies an HDF5 object*

Description

Copies an HDF5 object

Usage

```
H5Ocopy(h5loc, name, h5loc_dest, name_dest, obj_cpy_pl = NULL, lcpl = NULL)
```

Arguments

`h5loc`	An object of class H5IdComponent representing an open HDF5 object where the source object should be copied from.
`name`	Character vector of length 1, giving the name of the source object to be copied.
`h5loc_dest`	An object of class H5IdComponent representing an open HDF5 object where the new copy should be created.
`name_dest`	Character vector of length 1, giving the name of the new object to be created.
`obj_cpy_pl, lcpl`	H5IdComponent objects representing object copy and link creation property lists respectively. If left as `NULL` the default values for these will be used.

Examples

```
## Create a temporary copy of an example file check the contents
example_file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")
file.copy(example_file, tempdir())
h5_file <- file.path(tempdir(), "h5ex_t_array.h5")
h5ls(h5_file)

## open the example file and create a new, empty, file
fid1 <- H5Fopen(h5_file)
h5_file2 <- tempfile(fileext = ".h5")
fid2 <- H5Fcreate(h5_file2)

## We can copy a dataset inside the same file
H5Ocopy(h5loc = fid1, name = "DS1", h5loc_dest = fid1, name_dest = "DS2")
## Or to a different file
H5Ocopy(h5loc = fid1, name = "DS1", h5loc_dest = fid2, name_dest = "DS1_copy")
```

```
## if we want to create a new group hierarchy we can use a link creation property list
lcpl <- H5Pcreate("H5P_LINK_CREATE")
H5Pset_create_intermediate_group(lcpl, create_groups = TRUE)
H5Ocopy(
  h5loc = fid1, name = "DS1", h5loc_dest = fid2,
  name_dest = "/foo/baa/DS1_nested", lcpl = lcpl
)

## tidy up
H5Pclose(lcpl)
H5Fclose(fid1)
H5Fclose(fid2)

## Check we now have groups DS1 and DS2 in the original file
h5ls(h5_file)
## Check we have a copy of DS1 at the root and nests in the new file
h5ls(h5_file2)
```

H5Oget_info — *Retrieves the metadata for an HDF5 object specified by an identifier.*

Description

Retrieves the metadata for an HDF5 object specified by an identifier.

Usage

```
H5Oget_info(h5loc)
```

Arguments

`h5loc`	An object of class H5IdComponent representing an open HDF5 dataset or group.

Examples

```
## Create a temporary copy of an example file check the contents
example_file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")

## open the example file, root group, and DS1 dataset
fid <- H5Fopen(example_file)
gid <- H5Gopen(fid, "/")
did <- H5Dopen(fid, "/DS1")

## List the available object information for both groups and datasets
H5Oget_info(h5loc = gid)

H5Oget_info(h5loc = did)
```

```
## close open handles
h5closeAll(did, gid, fid)
```

`H5Oget_num_attrs` *Find the number of attributes associated with an HDF5 object*

Description

Find the number of attributes associated with an HDF5 object

Usage

```
H5Oget_num_attrs(h5obj)

H5Oget_num_attrs_by_name(h5loc, name)
```

Arguments

`h5obj`	An object of class H5IdComponent representing a H5 object identifier (file, group, or dataset).
`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group).
`name`	The name of the object to be checked.

Details

These functions are not part of the standard HDF5 C API.

Value

Returns a vector of length 1 containing the number of attributes the specified object has.

`H5Olink` *Create a hard link to an object in an HDF5 file*

Description

Create a hard link to an object in an HDF5 file

Usage

```
H5Olink(h5obj, h5loc, newLinkName, lcpl = NULL, lapl = NULL)
```

Arguments

`h5obj`	An object of class H5IdComponent representing the object to be linked to.
`h5loc`	An object of class H5IdComponent representing the location at which the object is to be linked. Can represent a file, group, dataset, datatype or attribute.
`newLinkName`	Character string giving the name of the new link. This should be relative to `h5loc`.
`lcpl`, `lapl`	H5IdComponent objects representing link creation and link access property lists respectively. If left as `NULL` the default values for these will be used.

See Also

H5Gcreate_anon

Examples

```
## Create a temporary copy of an example file, and open it
example_file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")
file.copy(example_file, tempdir())
h5_file <- file.path(tempdir(), "h5ex_t_array.h5")
fid <- H5Fopen(h5_file)

## create a new group without a location in the file
gid <- H5Gcreate_anon(fid)

## create link to newly create group
## relative to the file identifier
H5Olink(h5obj = gid, h5loc = fid, newLinkName = "foo")

## tidy up
H5Gclose(gid)
H5Fclose(fid)

## Check we now have a "/foo" group
h5ls(h5_file)
```

`H5Oopen` *Open an object in an HDF5 file*

Description

Open an object in an HDF5 file

Usage

```
H5Oopen(h5loc, name)
```

Arguments

`h5loc`	An object of class H5IdComponent
`name`	Path to the object to be opened. This should be relative to `h5loc` rather than the file.

Value

An object of class H5IdComponent if the open operation was successful. `FALSE` otherwise.

See Also

`H5Oclose()`

Examples

```
h5File <- tempfile(pattern = "ex_H5O.h5")

# create an hdf5 file and write something
h5createFile(h5File)
h5createGroup(h5File, "foo")
B <- array(seq(0.1, 2.0, by = 0.1), dim = c(5, 2, 2))
h5write(B, h5File, "foo/B")

# reopen file and dataset and get object info
fid <- H5Fopen(h5File)
oid <- H5Oopen(fid, "foo")
H5Oget_num_attrs(oid)
H5Oclose(oid)
H5Fclose(fid)
```

`H5Pall_filters_avail` *Query dataset filter properties.*

Description

Return information about the filter pipeline applied to a dataset creation property list.

Usage

```
H5Pall_filters_avail(h5plist)

H5Pget_nfilters(h5plist)

H5Pget_filter(h5plist, idx)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.
`idx`	Integer of length 1. This argument selects which filter to return information about. Indexing is R-style 1-based.

Details

- `H5Pall_filters_avail()` checks whether all filters required to process a dataset are available to **rhdf5**. This can be required if reading files created with other HDF5 software.
- `H5Pget_nfilters()` returns the number of filters in the dataset chunk processing pipeline.
- `H5Pget_filter()` provides details of a specific filter in the pipeline. This includes the filter name and the parameters provided to it e.g. compression level.

`H5Pclose` *Close and release a property list*

Description

`H5Pclose()` terminates access to a property list. All property lists should be closed when they no longer need to be accessed. This frees resources used by the property list. Failing to call `H5Pclose()` can lead to memory leakage over time.

Usage

```
H5Pclose(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the property list to close.

`H5Pcopy` *Copy an existing property list to create a new property list*

Description

Copy an existing property list to create a new property list

Usage

```
H5Pcopy(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the property list to be copied.

`H5Pcreate` *Create a new HDF5 property list*

Description

Create a new HDF5 property list

Usage

```
H5Pcreate(type = h5default("H5P"), native = FALSE)
```

Arguments

`type`	A character name of a property list type. See `h5const("H5P")` for possible property list types.
`native`	Defunct! Doesn't achieve anything for property lists.

`H5Pfill_value_defined` *Determine whether a property list has a fill value defined*

Description

Determine whether a property list has a fill value defined

Usage

```
H5Pfill_value_defined(h5plist)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.

Details

Note that the return value for this function is slightly different from the C version. The C API provides three return types and can, in the case that a fill value is defined, differentiate whether the value is the HDF5 library default or has been set by the application.

Value

`TRUE` if the fill value is defined, `FALSE` if not. Will return `NULL` if there is a problem determining the status of the fill value.

`H5Pget_class` *Return the property list class identifier for a property list*

Description

Return the property list class identifier for a property list

Usage

```
H5Pget_class(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing any type of HDF5 property list.

`H5Pget_version` *Get version information for objects in a file creation property list*

Description

Get version information for objects in a file creation property list

Usage

```
H5Pget_version(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list

Value

Named integer vector

`H5Pobject_track_times` *Set whether to record timestamps for operations performed on an HDF5 object.*

Description

Set whether to record timestamps for operations performed on an HDF5 object.

Usage

```
H5Pset_obj_track_times(h5plist, track_times = TRUE)

H5Pget_obj_track_times(h5plist)
```

Arguments

`h5plist`	An H5IdComponent object representing an object creation property list.
`track_times`	`logical` specifying whether times associated with an object should recorded.

Details

Objects created using high-level **rhdf5** functions like `h5createDataset()` will have this setting turned off. This was done to ensure otherwise identical files returned the same md5 hash. This differs from the default setting in HDF5, which is for objects to record the times operations were performed on them.

`H5Pset_blosc` *Add the BLOSC filter to the chunk processing pipeline.*

Description

Add the BLOSC filter to the chunk processing pipeline.

Usage

```
H5Pset_blosc(h5plist, h5tid, method = 1L, level = 6L, shuffle = TRUE)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.
`h5tid`	HDF5 data type id
`method`	Integer defining which of the compression algorithms provided by BLOSC should be used. (See the details section for the mapping between integers and algorithms).
`level`	Compression level to be used by the selected algorithm.

`shuffle`	Logical defining whether the bit-shuffle algorithm should be used prior to compression. This makes use of the shuffle implementation provide by BLOSC, rather than the HDF5 version.

`H5Pset_bzip2` *Add the BZIP2 filter to the chunk processing pipeline.*

Description

Add the BZIP2 filter to the chunk processing pipeline.

Usage

```
H5Pset_bzip2(h5plist, level = 2L)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.
`level`	Compression level to be used by the selected algorithm.

`H5Pset_deflate` *Add the deflate compression filter to the chunk processing pipeline.*

Description

Valid values for the compression level range from 0 (no compression) to 9 (best compression, slowest speed). Note that applying this function with `level = 0` does not mean the filter is removed. It is still part of the filter pipeline, but no compression is performed. The filter will still need to be available on any system that reads a file created with this setting

Usage

```
H5Pset_deflate(h5plist, level)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.
`level`	Integer giving the compression level to use. Valid values are from 0 to 9.

H5Pset_fapl_ros3 *Set the read-only S3 virtual file driver*

Description

The read-only S3 virtual file driver can be used to read files hosted remotely on Amazon's S3 storage.

Usage

```
H5Pset_fapl_ros3(h5plist, s3credentials = NULL)
```

Arguments

`h5plist`	H5IdComponent object representing a file access property list.
`s3credentials`	Either `NULL` or a list of length 3 specifying the AWS access credentials (see details).

Details

To access files in a private Amazon S3 bucket you will need to provide three additional details: The AWS region where the files are hosted, your AWS access key ID, and your AWS secret access key. More information on how to obtain AWS access keys can be found at `https://docs.aws.amazon.com/general/latest/gr/aws-sec-cred-types.html#access-keys-and-secret-access-keys`. These are provided as a list to the `s3credentials` argument. If you are accessing public data this argument should be `NULL`.

Examples

```
## this doesn't work on the Bioconductor Mac build machine
## Not run:
pid <- H5Pcreate("H5P_FILE_ACCESS")
H5Pset_fapl_ros3(pid)
H5Pclose(pid)

## End(Not run)
```

H5Pset_filter *Add a filter to the dataset filter pipeline.*

Description

Add a filter to the dataset filter pipeline.

Usage

```
H5Pset_filter(h5plist, filter_id, is_mandatory = FALSE, cd_values)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.
`filter_id`	Integer of length 1, giving the ID of the filter to be used.
`is_mandatory`	Logical of length 1. Filters can be either optional or mandatory. If this argument is set to `FALSE` the filter won't be applied to a chunk in the case of failure, but the data will still be written. Setting to `TRUE` will result in a failure when writing the dataset if the filter fails for some reason.
`cd_values`	Integer vector giving parameters to be supplied to the filter. No guidance is given for the number of values supplied here, it is specific to each filter and the user is expected to know appropriate options for the requested filter.

`H5Pset_istore_k` *Get and set the 1/2 rank of an indexed storage B-tree*

Description

Get and set the 1/2 rank of an indexed storage B-tree

Usage

```
H5Pset_istore_k(h5plist, ik)

H5Pget_istore_k(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list
`ik`	chunked Storage B-tree 1/2 rank

`H5Pset_lzf` *Add the LZF filter to the chunk processing pipeline.*

Description

Add the LZF filter to the chunk processing pipeline.

Usage

```
H5Pset_lzf(h5plist, h5tid)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.
`h5tid`	HDF5 data type id

`H5Pset_nbit` *Add the N-Bit filter to the chunk processing pipeline.*

Description

Add the N-Bit filter to the chunk processing pipeline.

Usage

```
H5Pset_nbit(h5plist)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.

Value

Returns (invisibly) an integer vector of length 1. The only element of this vector will be non-negative if the filter was set successfully and negative otherwise.

`H5Pset_shared_mesg_index` *Get and set shared object header message index properties*

Description

Get and set shared object header message index properties

Usage

```
H5Pset_shared_mesg_index(
  h5plist,
  index_num,
  mesg_type_flags = h5default(type = "H5O_SHMESG_FLAG"),
  min_mesg_size
)

H5Pget_shared_mesg_index(h5plist, index_num)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list
`index_num`	Index being configured. Indices use C-style 0-based counting, so the first index will be numbered 0.
`mesg_type_flags`	Character specifying the types of messages that may be stored in this index. Valid values can be found with `h5const(type = "H5O_SHMESG_FLAG")`
`min_mesg_size`	Minimum message size

Value

`H5Pget_shared_mesg_index()` returns a list of length 2. The first element is the types of messages that may be stored in the index, the second element is the minimum message size.

`H5Pset_shared_mesg_nindexes` *Get and set the number of object header message indexes*

Description

Get and set the number of object header message indexes

Usage

```
H5Pset_shared_mesg_nindexes(h5plist, nindexes)

H5Pget_shared_mesg_nindexes(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list
`nindexes`	Number of shared object header message indexes to be available in files

---

`H5Pset_shared_mesg_phase_change` *Get and set threshold values for storage of shared object header message indexes*

Description

Get and set threshold values for storage of shared object header message indexes

Usage

```
H5Pset_shared_mesg_phase_change(h5plist, max_list, min_btree)

H5Pget_shared_mesg_phase_change(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list
`max_list`	Threshold above which storage shifts from list to B-tree
`min_btree`	Threshold below which storage reverts to list format

`H5Pset_shuffle` *Add the shuffle filter to the chunk processing pipeline.*

Description

Add the shuffle filter to the chunk processing pipeline.

Usage

```
H5Pset_shuffle(h5plist)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.

Value

Returns (invisibly) an integer vector of length 1. The only element of this vector will be non-negative if the filter was set successfully and negative otherwise.

`H5Pset_sizes` *Get and set the sizes of offsets and lengths used in an HDF5 file*

Description

Get and set the sizes of offsets and lengths used in an HDF5 file

Usage

```
H5Pset_sizes(h5plist, sizeof_addr, sizeof_size)

H5Pget_sizes(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list
`sizeof_addr`	Offset size in bytes
`sizeof_size`	Length size in bytes

`H5Pset_sym_k` *Get and set the size of the symbol table B-tree 1/2 rank and the leaf node 1/2 size*

Description

Get and set the size of the symbol table B-tree 1/2 rank and the leaf node 1/2 size

Usage

```
H5Pset_sym_k(h5plist, ik, lk)

H5Pget_sym_k(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list
`ik`	Symbol table B-tree 1/2 rank
`lk`	Symbol table leaf node 1/2 size

`H5Pset_szip` *Add the SZIP compression filter to the chunk processing pipeline.*

Description

Add the SZIP compression filter to the chunk processing pipeline.

Usage

```
H5Pset_szip(h5plist, options_mask, pixels_per_block)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset creation property list.
`options_mask, pixels_per_block`	Integer vectors of length 1, setting parameters of the SZIP algorithm. See `https://portal.hdfgroup.org/display/HDF5/H5P_SET_SZIP` for more details.

References

`https://portal.hdfgroup.org/display/HDF5/Szip+Compression+in+HDF+Products`

`H5Pset_userblock` *Get and set the user block size*

Description

Get and set the user block size

Usage

```
H5Pset_userblock(h5plist, size)

H5Pget_userblock(h5plist)
```

Arguments

`h5plist`	H5IdComponent object representing the file creation property list
`size`	of the user block in bytes

`H5P_chunk` *Get and set the size of the chunks used to store a chunked layout dataset*

Description

Get and set the size of the chunks used to store a chunked layout dataset

Usage

```
H5Pset_chunk(h5plist, dim)

H5Pget_chunk(h5plist)
```

Arguments

`h5plist`	An object of class H5IdComponent representing a dataset creation property list.
`dim`	The chunk size used to store the dataset. This argument should be an integer vector of the same length as the number of dimensions of the dataset the dataset creation property list will be applied to.

Details

Note that a necessary side effect of running this function is that the layout of the dataset will be changes to `H5D_CHUNKED` if it is not already set to this.

See Also

`H5Pset_layout()`

`H5P_chunk_cache` *Set parameters for the raw data chunk cache*

Description

Set parameters for the raw data chunk cache

Usage

```
H5Pset_chunk_cache(h5plist, rdcc_nslots, rdcc_nbytes, rdcc_w0)
```

Arguments

`h5plist`	Object of class H5IdComponent representing a dataset access property list.
`rdcc_nslots`	Integer defining the number of chunk slots in the raw data chunk cache for this dataset.
`rdcc_nbytes`	Integer setting the total size of the raw data chunk cache for this dataset in bytes. In most cases increasing this number will improve performance, as long as you have enough free memory. The default size is 1 MB
`rdcc_w0`	Numeric value defining the chunk preemption policy. Must be between `0` and `1` inclusive.

`H5P_create_intermediate_group` *Get and set whether to create missing intermediate groups*

Description

Get and set whether to create missing intermediate groups

Usage

```
H5Pset_create_intermediate_group(h5plist, create_groups = TRUE)

H5Pget_create_intermediate_group(h5plist)
```

Arguments

`h5plist`	An object of class H5IdComponent representing a link creation property list.
`create_groups`	A logical of length 1 specifying whether missing groups should be created when a new object is created. Default is `TRUE`.

Examples

```
pid <- H5Pcreate("H5P_LINK_CREATE")

## by default intermediate groups are not created
H5Pget_create_intermediate_group(pid)

## Change the setting so groups will be created

H5Pget_create_intermediate_group(pid)

## tidy up
H5Pclose(pid)
```

`H5P_fill_time` *Set the time when fill values are written to a dataset*

Description

Set the time when fill values are written to a dataset

Usage

```
H5Pset_fill_time(h5plist, fill_time = h5default("H5D_FILL_TIME"))

H5Pget_fill_time(h5plist)
```

Arguments

`h5plist`	An object of class H5IdComponent representing a dataset creation property list.
`fill_time`	When the fill values should be written. Possible options can be listed with `h5const("H5D_FILL_TIME")`.

`H5P_fill_value` *Set the fill value for an HDF5 dataset*

Description

`H5Pset_fill_value` sets the fill value for a dataset in the dataset creation property list.

Usage

```
H5Pset_fill_value(h5plist, value)
```

Arguments

`h5plist`	An object of class H5IdComponent representing a dataset creation property list.
`value`	The default fill value of the dataset. A vector of length 1.

See Also

H5P_fill_time,H5Pfill_value_defined

`H5P_layout` *Get and set the type of storage used to store the raw data for a dataset*

Description

Possible options for the `layout` argument are:

- `H5D_COMPACT`
- `H5D_CONTIGUOUS`
- `H5D_CHUNKED`
- `H5D_VIRTUAL`

Usage

```
H5Pset_layout(h5plist, layout = h5default("H5D"))

H5Pget_layout(h5plist)
```

Arguments

`h5plist`	An object of class H5IdComponent representing a dataset creation property list.
`layout`	A character giving the name of a dataset layout type.

Details

The names of the layout types can also be obtained via `h5const("H5D")`.

H5P_libver_bounds *Control the range of HDF5 library versions that will be compatible with a file.*

Description

Control the range of HDF5 library versions that will be compatible with a file.

Usage

```
H5Pset_libver_bounds(
  h5plist,
  libver_low = "H5F_LIBVER_EARLIEST",
  libver_high = "H5F_LIBVER_LATEST"
)

H5Pget_libver_bounds(h5plist)
```

Arguments

h5plist	H5IdComponent object representing a file access property list.
libver_low, libver_high	Define the earliest and latest versions of the HDF5 library that will be used when writing object in the file.

H5R *H5R - References to objects and regions*

Description

The H5R functions can be used for creating or working with references to specific objects and data regions in an HDF5 file.

Author(s)

Mike Smith

Examples

```
library(rhdf5)

## first we'll create a file with a group named "foo" and a
## 1-dimensional dataset named "baa" inside that group.
file_name <- tempfile(fileext = ".h5")
h5createFile(file_name)
h5createGroup(file = file_name, group = "/foo")
```

```
h5write(1:100, file = file_name, name = "/foo/baa")


fid <- H5Fopen(file_name)
ref_to_group <- H5Rcreate(fid, name = "/foo")
ref_to_dataset <- H5Rcreate(fid, name = "/foo/baa")
two_refs <- c(ref_to_group, ref_to_dataset)
two_refs

## the size of this dataspace is the number of object references
## we want to store
sid <- H5Screate_simple(2)
tid <- H5Tcopy(dtype_id = "H5T_STD_REF_OBJ")
did <- H5Dcreate(fid, name = "object_refs", dtype_id = tid, h5space = sid)
H5Dwrite(did, two_refs)
H5Dclose(did)
H5Sclose(sid)
H5Fclose(fid)
```

H5Rcreate *Create a reference*

Description

Creates a reference to an object or dataset selection inside an HDF5 file.

Usage

```
H5Rcreate(h5loc, name, ref_type = "H5R_OBJECT", h5space = NULL)
```

Arguments

h5loc	An `H5IdComponent` object representing the location to be pointed to by the created reference.
name	Character string giving the name of the object to be referenced, relative to the location given by `h5loc`.
ref_type	The type of reference to create. Accepts either `H5R_OBJECT` or `H5R_DATASET_REGION`.
h5space	An object of class `H5IdComponent` representing a dataspace with a selection set. This argument is only used if creating a reference to a dataset region, and will be ignored otherwise.

Value

An H5Ref object storing the reference.

`H5Rdereference` *Open a reference object.*

Description

Given a reference and the file to which that reference applies, `H5Rdeference()` will open the reference object and return an identifier.

Usage

```
H5Rdereference(ref, h5loc)
```

Arguments

`ref`	`H5ref` object containing the reference to be opened.
`h5loc`	An `H5IdComponent` object representing the file containing the referenced object.

Details

If `ref` contains more than one reference, only the first reference will be used. It must be subset with `[` if one of the other stored references should be opened.

Value

An object of class `H5IdComponent` representing the opened object referenced by `ref`. This should be closed with the appropriate function e.g. `H5Dclose()`, `H5Oclose()`, etc. when no longer needed.

`h5readTimestamps` *Read the time stamps associated with an HDF5 group or dataset.*

Description

Read the time stamps associated with an HDF5 group or dataset.

Usage

```
h5readTimestamps(file, name)
```

Arguments

`file`	Character vector of length 1, giving the path to the HDF5 file
`name`	Path within the HDF5 file to the object whose attributes should be read. The datasets present in `file` can be listed with the function `h5ls`.

Details

All timestamps are returned in the UTC timezone. HDF5 objects can have between 0 and 4 timestamps set, depending on the property lists provided when they are created or accessed. Timestamps that are not tracked will be returned as the UNIX epoch `1970-01-01 UTC`.

Value

A named list of length 4 containing the timestamps on the object. The timestamps themselves are `POSIXct` objects (see `base::DateTimeClasses()`).

Examples

```
# example file
example_file <- system.file("testfiles", "h5ex_t_array.h5", package = "rhdf5")

## read timestamps on a group
h5readTimestamps(example_file, name = "/")

## read timestamps on a datasets
h5readTimestamps(example_file, name = "/DS1")
```

`H5Ref-class` *An S4 class representing H5 references.*

Description

A class representing one or more HDF5 references.

Usage

```
## S4 method for signature 'H5Ref'
show(object)

## S4 method for signature 'H5Ref'
length(x)

## S4 method for signature 'H5Ref'
c(x, ...)

## S4 method for signature 'H5Ref'
x[i]
```

Arguments

`object`	Object of class `H5Ref`
`x`	An `H5Ref` object.
`...`	Additional `H5Ref` objects to be combined with `x`.
`i`	Integer vector giving the indices of references to select.

Details

The length of the `val` slot is dependent on both the number and type of references stored in the object. `H5R_OBJECT` references are stored in 8 bytes, while `H5R_DATASET_REGION` references require 12 bytes. The length of `val` will then be a multiple of 8 or 12 respectively. This also means that references of different types cannot be combined in a single object.

Methods (by generic)

- `show(H5Ref)`: Print details of the object to screen.
- `length(H5Ref)`: Return the number of references stored in an `H5Ref` object.
- `c(H5Ref)`: Combine two or more `H5Ref` objects. Objects must all contain the same type of reference, either `H5R_OBJECT` or `H5R_DATASET_REFERENCE`.
- `[`: Subset an `H5Ref` object.

Slots

`val` `raw` vector containing the byte-level representation of each reference.

`type` `integer` of length 1, which maps to either `H5R_OBJECT` or `H5R_DATASET_REGION`.

`H5Rget_name` *Return the name of the object that a reference points to*

Description

Return the name of the object that a reference points to

Usage

```
H5Rget_name(ref, h5loc)
```

Arguments

`ref`	`H5ref` object containing the reference to be queried.
`h5loc`	An `H5IdComponent` object representing the file containing the referenced object.

Value

Character string of length 1 giving the name of the referenced object.

`H5Rget_obj_type` *Identify the type of object that a reference points to*

Description

Identify the type of object that a reference points to

Usage

```
H5Rget_obj_type(ref, h5loc)
```

Arguments

`ref`	`H5ref` object containing the reference to be queried.
`h5loc`	An `H5IdComponent` object representing the file containing the referenced object.

Value

Character string of length 1 identifying the object type. Valid return values are: `"GROUP"`, `"DATASET"`, and `"NAMED_DATATYPE"`.

`H5Rget_region` *Return selection for a reference to dataset region*

Description

Given a dataset region reference, this function will return the dataspace and selection required to read the data points indicated by the reference.

Usage

```
H5Rget_region(ref, h5loc)
```

Arguments

`ref`	An object of class `H5Ref`. This function is only valid for reference of type `H5R_DATASET_REGION`, and not `H5R_OBJECT`.
`h5loc`	An `H5IdComponent` object representing the file containing the referenced object.

Value

An object of class `H5IdComponent` representing the dataspace of the dataset that `ref` points to. The dataspace will have the selection set that matches the selection pointed to by `ref`. This should be closed using `H5Sclose()` when no longer required.

H5Sclose — *Close and release a dataspace*

Description

Close and release a dataspace

Usage

```
H5Sclose(h5space)
```

Arguments

`h5space`	Object of class H5IdComponent representing the dataspace to be closed.

See Also

`H5Screate()`

H5Scombine_hyperslab — *Perform operation between an existing selection and an another hyperslab definition.*

Description

Combines a hyperslab selection specified by `start`, `stride`, `count` and `block` arguments with the current selection for the dataspace represented by `h5space`.

Usage

```
H5Scombine_hyperslab(
  h5space,
  op = h5default("H5S_SELECT"),
  start = NULL,
  stride = NULL,
  count = NULL,
  block = NULL
)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.
`op`	Character string defined the operation used to join the two dataspaces. See `h5const("H5S_SELECT")` for the list of available options.
`start`, `stride`, `count`, `block`	Integer vectors, each with length equal to the rank of the dataspace. These parameters define the new hyperslab to select.

Value

An H5IdComponent object representing a new dataspace with the generated selection.

See Also

`H5Scombine_select()`, `H5Sselect_hyperslab()`

Examples

```
## create a 1 dimensional dataspace
sid_1 <- H5Screate_simple(dims = 20)

## select a single block of 5 points in sid_1
## this is equivalent to [11:16] in R syntax
H5Sselect_hyperslab(sid_1,
  start = 11, stride = 1,
  block = 5, count = 1
) #

## combine the existing selection with a new
## selection consisting of 2 blocks each of 1 point
## equivalent to [c(3,5)] in R syntax
sid_2 <- H5Scombine_hyperslab(sid_1,
  op = "H5S_SELECT_OR",
  start = 3, stride = 2,
  block = 1, count = 2
)

## confirm we have selected 5 in our original dataspace
## and 7 points in the newly created dataspace
H5Sget_select_npoints(sid_1)
H5Sget_select_npoints(sid_2)

## tidy up
H5Sclose(sid_1)
H5Sclose(sid_2)
```

`H5Scombine_select` *Combine two selections*

Description

Combine two selections

Usage

```
H5Scombine_select(h5space1, op = h5default("H5S_SELECT"), h5space2)
```

Arguments

`h5space1, h5space2`
: H5IdComponent objects representing a dataspaces.

`op`
: Character string defined the operation used to join the two dataspaces. See `h5const("H5S_SELECT")` for the list of available options.

Value

Returns an H5IdComponent object representing a new dataspace. The new dataspace will have the same extent as `h5space1` with the hyperslab selection being the result of combining the selections of `h5space1` and `h5space2`.

See Also

`H5Scombine_hyperslab()`

Examples

```
## create two 1 dimensional dataspaces
## of different sizes
sid_1 <- H5Screate_simple(dims = 20)
sid_2 <- H5Screate_simple(dims = 10)

## select a single block of 5 points in sid_1
## this is equivalent to [11:16] in R syntax
H5Sselect_hyperslab(sid_1,
  start = 11, stride = 1,
  block = 5, count = 1
)

## select 2 blocks of 1 point from sid_2
## equivalent to [c(3,5)] in R syntax
H5Sselect_hyperslab(sid_2,
  start = 3, stride = 2,
  block = 1, count = 2
)

## confirm we have select 5 and 2 points resepectively
H5Sget_select_npoints(sid_1)
H5Sget_select_npoints(sid_2)

## combine the two dataset selections keeping points that
## are in one or both of the selections
sid_3 <- H5Scombine_select(sid_1, "H5S_SELECT_OR", sid_2)

## extent of the new dataset is the same as sid_1
sid_3
## confirm the selection contains 7 points
H5Sget_select_npoints(sid_3)

## tidy up
```

```
H5Sclose(sid_1)
H5Sclose(sid_2)
H5Sclose(sid_3)
```

H5Scopy *Create a copy of a dataspace*

Description

`H5S_copy()` creates an exact copy of a given dataspace.

Usage

```
H5Scopy(h5space)
```

Arguments

`h5space`	Object of class H5IdComponent representing the dataspace to be copied.

Value

If the copying is successful returns an object of class H5IdComponent representing the new dataspace. Otherwise returns `FALSE`.

H5Screate *Create a new dataspace of a specified type*

Description

Create a new dataspace of a specified type

Usage

```
H5Screate(type = h5default("H5S"), native = FALSE)
```

Arguments

`type`	The type of dataspace to create. See `h5const("H5S")` for possible types.
`native`	An object of class `logical`. If `TRUE`, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`.

Value

Returns an object of class H5IdComponent representing a dataspace.

See Also

H5Screate_simple

`H5Screate_simple` *Create a simple dataspace*

Description

Create a simple dataspace

Usage

```
H5Screate_simple(dims, maxdims, native = FALSE)
```

Arguments

`dims`	A numeric vector defining the initial dimensions of the dataspace. The length of `dims` determines the rank of the dataspace.
`maxdims`	A numeric vector with the same length length as `dims`. Specifies the upper limit on the size of the dataspace dimensions. Only needs to be specified if this is different from the values given to `dims`.
`native`	An object of class `logical`. If `TRUE`, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`.

Value

Returns an object of class H5IdComponent representing a dataspace.

See Also

H5Screate

`H5Sget_select_npoints` *Find the number of elements in a dataspace selection*

Description

Find the number of elements in a dataspace selection

Usage

```
H5Sget_select_npoints(h5space)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.

`H5Sget_simple_extent_dims` *Find the size of a dataspace*

Description

Find the size of a dataspace

Usage

```
H5Sget_simple_extent_dims(h5space)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.

`H5Sis_simple` *Determine whether a dataspace is a simple dataspace*

Description

In HDF5 a dataspace is considered "simple" if it represents a regular N-dimensional array of points. Currently (HDF 1.10.7) all dataspaces are simple. Support for complex dataspaces is planned for future HDF versions.

Usage

```
H5Sis_simple(h5space)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.

`H5Sselect_all` *Set the selection region of a dataspace to include all elements*

Description

Set the selection region of a dataspace to include all elements

Usage

```
H5Sselect_all(h5space)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.

`H5Sselect_hyperslab` *Perform operation between an existing selection and an another hyperslab definition.*

Description

Combines a hyperslab selection specified by `start`, `stride`, `count` and `block` arguments with the current selection for the dataspace represented by `h5space`.

Usage

```
H5Sselect_hyperslab(
  h5space,
  op = h5default("H5S_SELECT"),
  start = NULL,
  stride = NULL,
  count = NULL,
  block = NULL
)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.
`op`	Character string defined the operation used to join the two dataspaces. See `h5const("H5S_SELECT")` for the list of available options.
`start`, `stride`, `count`, `block`	Integer vectors, each with length equal to the rank of the dataspace. These parameters define the new hyperslab to select.

Details

`H5Sselect_hyperslab` is similar to, but subtly different from, `H5Scombine_hyperslab()`. The former modifies the selection of the dataspace provided in the `h5space` argument, while the later returns a new dataspace with the combined selection.

Examples

```
## create a 1 dimensional dataspace
sid_1 <- H5Screate_simple(dims = 20)

## select a single block of 5 points in sid_1
## this is equivalent to [11:16] in R syntax
H5Sselect_hyperslab(sid_1,
  start = 11, stride = 1,
  block = 5, count = 1
)

## confirm we have selected 5 in our original dataspace
```

```
H5Sget_select_npoints(sid_1)

## combine the existing selection with a new
## selection consisting of 2 blocks each of 1 point
## equivalent to [c(3,5)] in R syntax
H5Sselect_hyperslab(sid_1,
  op = "H5S_SELECT_OR",
  start = 3, stride = 2,
  block = 1, count = 2
)

## The dataspace now has 7 points selected
H5Sget_select_npoints(sid_1)

## tidy up
H5Sclose(sid_1)
```

H5Sselect_index *Select elements of a dataspace using R-style indexing*

Description

Combines a hyperslab selection specified by `start`, `stride`, `count` and `block` arguments with the current selection for the dataspace represented by `h5space`.

Usage

```
H5Sselect_index(h5space, index)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.
`index`	A list of integer indices. The length of the list corresponds to the number of dimensions of the HDF5 array. If a list element is `NULL`, all elements of the respective dimension are selected.

Details

`H5Sselect_hyperslab` is similar to, but subtly different from, `H5Scombine_hyperslab()`. The former modifies the selection of the dataspace provided in the `h5space` argument, while the later returns a new dataspace with the combined selection.

Examples

```
## create a 1 dimensional dataspace
sid <- H5Screate_simple(c(10, 5, 3))

## Select elements that lie in in the rows 1-3, columns 2-4,
## and the entire 3rd dimension
H5Sselect_index(sid, list(1:3, 2:4, NULL))

## We can check the number of selected points.
## This should be 27 (3 * 3 * 3)
H5Sget_select_npoints(sid)

## always close dataspaces after usage to free resources
H5Sclose(sid)
```

`H5Sselect_none` *Set the selection region of a dataspace to include no elements*

Description

Set the selection region of a dataspace to include no elements

Usage

```
H5Sselect_none(h5space)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.

`H5Sselect_valid` *Check that a selection is valid*

Description

Check that a selection is valid

Usage

```
H5Sselect_valid(h5space)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.

`H5Sset_extent_simple` *Set the size of a dataspace*

Description

Set the size of a dataspace

Usage

```
H5Sset_extent_simple(h5space, dims, maxdims)
```

Arguments

`h5space`	H5IdComponent object representing a dataspace.
`dims`	Dimension of the dataspace. This argument is similar to the dim attribute of an array.
`maxdims`	Maximum extension of the dimension of the dataset in the file. If not provided, it is set to `dims`. When viewing the HDF5 dataset with other software (e.g. HDFView), the dimensions appear in inverted order, because the fastest changing dimension in R is the first one, and in C it's the last one.

`H5Sunlimited` *Retrieve value for* `H5S_UNLIMITED` *constant*

Description

The value for `H5S_UNLIMITED` can be provided to the `maxdims` argument of H5Screate_simple to indicate that the maximum size of the corresponding dimension is unlimited.

Usage

```
H5Sunlimited()
```

See Also

H5Screate_simple

`H5Tcopy` *Copy an existing datatype*

Description

Copy an existing datatype

Usage

```
H5Tcopy(dtype_id = h5default(type = "H5T"))
```

Arguments

`dtype_id`	Datatype to copy. Can either be a character specifying a predefined HDF5 datatype (see `h5const("H5T")` for valid options) or the ID of an already created datatype.

`H5Tis_variable_str` *Determine whether a datatype is a variable length string*

Description

Determine whether a datatype is a variable length string

Usage

```
H5Tis_variable_str(dtype_id)
```

Arguments

`dtype_id`	ID of HDF5 datatype to query.

`H5T_cset` *Retrieve or set the character set to be used in a string datatype.*

Description

Retrieve or set the character set to be used in a string datatype.

Usage

```
H5Tset_cset(dtype_id, cset = "ASCII")

H5Tget_cset(dtype_id)
```

Arguments

`dtype_id`	ID of HDF5 datatype to query or modify.
`cset`	Encoding to use for string types. Valid options are 'ASCII' and 'UTF-8'.

`H5T_enum` *Create or modify an HDF5 enum datatype*

Description

Create or modify an HDF5 enum datatype

Usage

```
H5Tenum_create(dtype_id = "H5T_NATIVE_INT")

H5Tenum_insert(dtype_id, name, value)
```

Arguments

`dtype_id`	ID of HDF5 datatype to work with. For `H5Tenum_create`, this is the identifier of the base data type, and must be an integer e.g. `H5T_NATIVE_INT`. For `H5Tenum_insert` this will be a datatype identifier created by `H5Tenum_create`.
`name`	The name of a the new enum member. This is analogous to a "level" in an R factor.
`value`	The value of the new member. Must be compatible with the base datatype defined by `dtype_id`.

Value

- `H5Tinsert_enum()` returns an character representing the H5 identifier of the new datatype.
- `H5Tset_precision()` is called for its side-effect of modifying the existing datatype. It will invisibly return `TRUE` if this is successful `FALSE` if not.

Examples

```
tid <- H5Tenum_create(dtype_id = "H5T_NATIVE_UCHAR")
H5Tenum_insert(tid, name = "TRUE", value = 1L)
H5Tenum_insert(tid, name = "FALSE", value = 0L)
```

H5T_ops — *Get details of HDF5 data types*

Description

Get details of HDF5 data types

Usage

```
H5Tget_class(dtype_id)

H5Tget_nmembers(dtype_id)
```

Arguments

`dtype_id`	ID of HDF5 datatype to work with. Normally created with a function like `H5Tcopy` or `H5Tenum_create`.

Value

- `H5Tget_class()` returns an character vector of length 1 giving the class of the data type.
- `H5Tget_nmembers()` returns the number of members in the given datatype. Will fail with an error if the supplied datatype is not of type `H5T_COMPUND` or `H5T_ENUM`.

Examples

```
## create an enum datatype with two entries
tid <- H5Tenum_create(dtype_id = "H5T_NATIVE_UCHAR")
H5Tenum_insert(tid, name = "TRUE", value = 1L)
H5Tenum_insert(tid, name = "FALSE", value = 0L)

H5Tget_class(tid)
H5Tget_nmembers(tid)
```

`H5T_precision` *Retrieve or set the precision of an HDF5 datatype*

Description

Retrieve or set the precision of an HDF5 datatype

Usage

```
H5Tset_precision(dtype_id, precision)

H5Tget_precision(dtype_id)
```

Arguments

`dtype_id`	ID of HDF5 datatype to set precision of.
`precision`	The number of bytes of precision for the datatype.

Value

- `H5Tget_precision()` returns an integer giving the number of significant bits used by the given datatype.
- `H5Tset_precision()` is call for its side-effect of modifying the precision of a datatype. It will invisibly return `TRUE` if this is successful and will stop with an error if the operation fails.

`H5T_size` *Retrieve or set the type of padding used by string datatype*

Description

Retrieve or set the type of padding used by string datatype

Usage

```
H5Tset_size(dtype_id = h5default(type = "H5T"), size)

H5Tget_size(dtype_id)
```

Arguments

`dtype_id`	ID of HDF5 datatype to query or modify.
`size`	The new datatype size in bytes.

H5T_strpad *Retrieve or set the type of padding used by string datatype*

Description

Retrieve or set the type of padding used by string datatype

Usage

```
H5Tset_strpad(dtype_id, strpad = "NULLPAD")

H5Tget_strpad(dtype_id)
```

Arguments

dtype_id	ID of HDF5 datatype to query or modify.
strpad	Character vector of length 1 specifying the type of padding to use. Valid options are NULLTERM, NULLPAD and SPACEPAD.

h5version *Print the rhdf5 and libhdf5 version numbers*

Description

Returns the version number of the Bioconductor package rhdf5 and the C-library libhdf5.

Usage

```
h5version()
```

Value

A list of major, minor and release number.

Author(s)

Bernd Fischer, Mike L. Smith

Examples

```
h5version()
```

`H5Zfilter_avail` *Determine whether a filter is available on this system*

Description

Determine whether a filter is available on this system

Usage

```
H5Zfilter_avail(filter_id)
```

Arguments

`filter_id`	Integer representing the ID of the filter to be checked.

`h5_createAttribute` *Create HDF5 attribute*

Description

R function to create an HDF5 attribute and defining its dimensionality.

Usage

```
h5createAttribute(
  obj,
  attr,
  dims,
  maxdims = dims,
  file,
  storage.mode = "double",
  H5type = NULL,
  size = NULL,
  encoding = NULL,
  native = FALSE
)
```

Arguments

`obj`	The name (character) of the object the attribute will be attached to. For advanced programmers it is possible to provide an object of class H5IdComponent representing a H5 object identifier (file, group, dataset). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()`, `H5Dcreate()`, `H5Dopen()` to create an object of this kind.
`attr`	Name of the attribute to be created.

`dims`	The dimensions of the attribute as a numeric vector. If `NULL`, a scalar dataspace will be created instead.
`maxdims`	The maximum extension of the attribute.
`file`	The filename (character) of the file in which the dataset will be located. For advanced programmers it is possible to provide an object of class H5IdComponent representing an H5 location identifier. See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind. The `file` argument is not required, if the argument `obj` is of type `H5IdComponent`.
`storage.mode`	The storage mode of the data to be written. Can be obtained by `storage.mode(mydata)`.
`H5type`	Advanced programmers can specify the datatype of the dataset within the file. See `h5const("H5T")` for a list of available datatypes. If `H5type` is specified the argument `storage.mode` is ignored. It is recommended to use `storage.mode`
`size`	The maximum string length when `storage.mode='character'`. If this is specified, HDF5 stores each string of `attr` as fixed length character arrays. Together with compression, this should be efficient. If this argument is set to `NULL`, HDF5 will instead store variable-length strings.
`encoding`	The encoding of the string data type i.e. when `storage.mode = 'character'`. Valid options are "ASCII" and "UTF-8".
`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

Details

Creates a new attribute and attaches it to an existing HDF5 object. The function will fail, if the file doesn't exist or if there exists already another attribute with the same name for this object.

You can use `h5writeAttribute()` immediately. It will create the attribute for you.

Value

Returns TRUE is attribute was created successfully and FALSE otherwise.

Author(s)

Bernd Fischer

References

`https://portal.hdfgroup.org/display/HDF5`

See Also

`h5createFile()`, `h5createGroup()`, `h5createDataset()`, `h5read()`, `h5write()`, rhdf5

Examples

```
h5File <- tempfile(pattern = "ex_createAttribute.h5")
h5createFile(h5File)
h5write(1:1, h5File, "A")
fid <- H5Fopen(h5File)
did <- H5Dopen(fid, "A")
h5createAttribute(did, "time", c(1, 10))
H5Dclose(did)
H5Fclose(fid)
```

`h5_createDataset` *Create HDF5 dataset*

Description

R function to create an HDF5 dataset and defining its dimensionality and compression behaviour.

Usage

```
h5createDataset(
  file,
  dataset,
  dims,
  maxdims = dims,
  storage.mode = "double",
  H5type = NULL,
  size = NULL,
  encoding = NULL,
  chunk = dims,
  fillValue,
  level = 6,
  filter = "gzip",
  shuffle = TRUE,
  native = FALSE
)
```

Arguments

`file`	The filename (character) of the file in which the dataset will be located. For advanced programmers it is possible to provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`dataset`	Name of the dataset to be created. The name can contain group names, e.g. 'group/dataset', but the function will fail, if the group does not yet exist.

`dims`	The dimensions of the array as they will appear in the file. Note, the dimensions will appear in inverted order when viewing the file with a C-program (e.g. HDFView), because the fastest changing dimension in R is the first one, whereas the fastest changing dimension in C is the last one.
`maxdims`	The maximum extension of the array. Use `H5Sunlimited()` to indicate an extensible dimension.
`storage.mode`	The storage mode of the data to be written. Can be obtained by `storage.mode(mydata)`.
`H5type`	Advanced programmers can specify the datatype of the dataset within the file. See `h5const("H5T")` for a list of available datatypes. If `H5type` is specified the argument `storage.mode` is ignored. It is recommended to use `storage.mode`
`size`	For `storage.mode='character'` the maximum string length to use. The default value of `NULL` will result in using variable length strings. See the details for more information on this option.
`encoding`	The encoding of the string data type. Valid options are "ASCII" or "UTF-8".
`chunk`	The chunk size used to store the dataset. It is an integer vector of the same length as `dims`. This argument is usually set together with a compression property (argument `level`).
`fillValue`	Standard value for filling the dataset. The storage.mode of value has to be convertible to the dataset type by HDF5.
`level`	The compression level used. An integer value between 0 (no compression) and 9 (highest and slowest compression).
`filter`	Character defining which compression filter should be applied to the chunks of the dataset. See the Details section for more information on the options that can be provided here.
`shuffle`	Logical defining whether the byte-shuffle algorithm should be applied to data prior to compression.
`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

Details

Creates a new dataset in an existing HDF5 file. The function will fail if the file doesn't exist or if there exists already another dataset with the same name within the specified file.

The `size` argument is only used when `storage.mode = 'character'`. When storing strings HDF5 can use either a fixed or variable length datatype. Setting `size` to a positive integer will use fixed length strings where `size` defines the length. **rhdf5** writes null padded strings by default and so to avoid data loss the value provided here should be the length of the longest string. Setting `size = NULL` will use variable length strings. The choice is probably dependent on the nature of the strings you're writing. The principle difference is that a dataset of variable length strings will not be compressed by HDF5 but each individual string only uses the space it requires, whereas in a fixed length dataset each string is of length uses `size`, but the whole dataset can be compressed. This explored more in the examples below.

The `filter` argument can take several options matching to compression filters distributed in either with the HDF5 library in **Rhdf5lib** or via the **rhdf5filters** package. The plugins available and the corresponding values for selecting them are shown below:

zlib: Ubiquitous deflate compression algorithm used in GZIP or ZIP files. All three options below achieve the same r
`"GZIP"`,
- `"ZLIB"`,
- `"DEFLATE"`

szip: Compression algorithm maintained by the HDF5 group. • `"SZIP"`

bzip2 • `"BZIP2"`

BLOSC meta compressor: As a meta-compressor BLOSC wraps several different compression algorithms. Each of t
`"BLOSC_BLOSCLZ"`
- `"BLOSC_LZ4"`
- `"BLOSC_LZ4HC"`
- `"BLOSC_SNAPPY"`
- `"BLOSC_ZLIB"`
- `"BLOSC_ZSTD"`

lzf • `"LZF"`

Disable: It is possible to write chunks without any compression applied. • `"NONE"`

Value

Returns (invisibly) `TRUE` if dataset was created successfully and `FALSE` otherwise.

Author(s)

Bernd Fischer, Mike L. Smith

See Also

`h5createFile()`, `h5createGroup()`, `h5read()`, `h5write()`

Examples

```
h5File <- tempfile(pattern = "_ex_createDataset.h5")
h5createFile(h5File)

# create dataset with compression
h5createDataset(h5File, "A", c(5, 8), storage.mode = "integer", chunk = c(5, 1), level = 6)

# create dataset without compression
h5createDataset(h5File, "B", c(5, 8), storage.mode = "integer")
h5createDataset(h5File, "C", c(5, 8), storage.mode = "double")

# create dataset with bzip2 compression
h5createDataset(h5File, "D", c(5, 8),
  storage.mode = "integer",
  chunk = c(5, 1), filter = "BZIP2", level = 6
)

# create a dataset of strings & define size based on longest string
ex_strings <- c("long", "longer", "longest")
h5createDataset(h5File, "E",
  storage.mode = "character", chunk = 3, level = 6,
  dims = length(ex_strings), size = max(nchar(ex_strings))
)


# write data to dataset
h5write(matrix(1:40, nr = 5, nc = 8), file = h5File, name = "A")
# write second column
h5write(matrix(1:5, nr = 5, nc = 1), file = h5File, name = "B", index = list(NULL, 2))
# write character vector
h5write(ex_strings, file = h5File, name = "E")

h5dump(h5File)

## Investigating fixed vs variable length string datasets

## create 1000 random strings with length between 50 and 100 characters
words <- vapply(
  X = ceiling(runif(n = 1000, min = 50, max = 100)),
  FUN = function(x) {
    paste(sample(letters, size = x, replace = TRUE),
      collapse = ""
    )
  },
  FUN.VALUE = character(1)
)

## create two HDF5 files
f1 <- tempfile()
f2 <- tempfile()
h5createFile(f1)
h5createFile(f2)

## create two string datasets
## the first is variable length strings, the second fixed at the length of our longest word
h5createDataset(f1, "strings",
  dims = length(words), storage.mode = "character",
  size = NULL, chunk = 25
)
h5createDataset(f2, "strings",
  dims = length(words), storage.mode = "character",
  size = max(nchar(words)), chunk = 25
)

## Write the data
h5write(words, f1, "strings")
h5write(words, f2, "strings")
```

```
## Check file sizes.
## In this example the fixed length string dataset is normally much smaller
file.size(f1)
file.size(f2)
```

---

h5_createFile *Create HDF5 file*

Description

R function to create an empty HDF5 file.

Usage

```
h5createFile(file)
```

Arguments

`file`	The filename of the HDF5 file.

Details

Creates an empty HDF5 file.

Value

Returns (invisibly) `TRUE` is file was created successfully and `FALSE` otherwise.

Author(s)

Bernd Fischer

See Also

`h5createGroup()`, `h5createDataset()`, `h5read()`, `h5write()`, rhdf5

Examples

```
h5File <- tempfile(pattern = "ex_createFile.h5")

h5createFile(h5File)

# create groups
h5createGroup(h5File, "foo")
h5createGroup(h5File, "foo/foobaa")

h5ls(h5File)
```

`h5_createGroup` *Create HDF5 group*

Description

Creates a group within an HDF5 file.

Usage

```
h5createGroup(file, group)
```

Arguments

`file`	The filename (character) of the file in which the dataset will be located. For advanced programmers it is possible to provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`group`	The name of the new group. The name can contain a hierarchy of groupnames, e.g. `"/group1/group2/newgroup"`, but the function will fail if the top level groups do not exists.

Details

Creates a new group within an HDF5 file.

Value

Returns TRUE is group was created successfully and FALSE otherwise.

Author(s)

Bernd Fischer

See Also

`h5createFile()`, `h5createDataset()`, `h5read()`, `h5write()`

Examples

```
h5File <- tempfile(pattern = "ex_createGroup.h5")
h5createFile(h5File)

# create groups
h5createGroup(h5File, "foo")
h5createGroup(h5File, "foo/foobaa")

h5ls(h5File)
```

`h5_delete` *Delete objects within a HDF5 file*

Description

Deletes the specified group or dataset from within an HDF5 file.

Usage

```
h5delete(file, name)
```

Arguments

`file`	The filename (character) of the file in which the object is located.
`name`	For `h5delete` the name of the object to be deleted. For `h5deleteAttribute` the name of the object to which the attribute belongs.

Author(s)

Mike Smith

`h5_deleteAttribute` *Delete attribute*

Description

Deletes an attribute associated with a group or dataset within an HDF5 file.

Usage

```
h5deleteAttribute(file, name, attribute)
```

Arguments

`file`	The filename (character) of the file in which the object is located.
`name`	The name of the object to which the attribute belongs.
`attribute`	Name of the attribute to be deleted.

Author(s)

Mike Smith

h5_dump — *Dump the content of an HDF5 file.*

Description

Dump the content of an HDF5 file.

Usage

```
h5dump(
  file,
  recursive = TRUE,
  load = TRUE,
  all = FALSE,
  index_type = h5default("H5_INDEX"),
  order = h5default("H5_ITER"),
  s3 = FALSE,
  s3credentials = NULL,
  ...,
  native = FALSE
)
```

Arguments

`file`	The filename (character) of the file in which the dataset will be located. You can also provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`recursive`	If `TRUE`, the content of the whole group hierarchy is listed. If `FALSE`, Only the content of the main group is shown. If a positive integer is provided this indicates the maximum level of the hierarchy that is shown.
`load`	If `TRUE` the datasets are read in, not only the header information. Note, that this can cause memory problems for very large files. In this case choose `load=FALSE` and load the datasets successively.
`all`	If `TRUE`, a longer list of information on each entry is provided.
`index_type`	See `h5const("H5_INDEX")` for possible arguments.
`order`	See `h5const("H5_ITER")` for possible arguments.
`s3`	Logical value indicating whether the file argument should be treated as a URL to an Amazon S3 bucket, rather than a local file path.
`s3credentials`	A list of length three, providing the credentials for accessing files in a private Amazon S3 bucket.
`...`	Arguments passed to `h5read()`
`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

Value

Returns a hierarchical list structure representing the HDF5 group hierarchy. It either returns the datasets within the list structure (`load=TRUE`) or it returns a `data.frame` for each dataset with the dataset header information (`load=FALSE`).

Author(s)

Bernd Fischer, Mike L. Smith

See Also

`h5ls()`

Examples

```
h5File <- tempfile(pattern = "ex_dump.h5")
h5createFile(h5File)

# create groups
h5createGroup(h5File, "foo")
h5createGroup(h5File, "foo/foobaa")

# write a matrix
B <- array(seq(0.1, 2.0, by = 0.1), dim = c(5, 2, 2))
attr(B, "scale") <- "liter"
h5write(B, h5File, "foo/B")

# list content of hdf5 file
h5dump(h5File)

# list content of an hdf5 file in a public S3 bucket

h5dump(file = "https://rhdf5-public.s3.eu-central-1.amazonaws.com/h5ex_t_array.h5", s3 = TRUE)
```

`h5_errorHandling` *Set how HDF5 error messages are displayed*

Description

Sets the options for handling HDF5 error messages in the R sessions.

Usage

```
h5errorHandling(type = "normal")
```

Arguments

`type`	'normal' (default) shows a one line error message in R. 'verbose' shows the whole HDF5 error message. 'suppress' suppresses the HDF5 error messages completely.

Value

Returns 0 if options are set successfully.

Author(s)

Bernd Fischer

See Also

rhdf5

Examples

```
h5errorHandling("normal")
```

`h5_FileLocking` *Test and set file locking for HDF5*

Description

HDF5 1.10 uses file locking by default. On some file systems this is not available, and the HDF5 library will throw an error if the user attempts to create or access a file located on such a file system. These functions help identify if file locking is available without throwing an error, and allow the locking to be disabled for the duration of the R session if needed.

Usage

```
h5testFileLocking(location)

h5disableFileLocking()

h5enableFileLocking()
```

Arguments

`location`	The name of a directory or file to test. If an existing directory is provided a temporary file will be created in this folder. If non-existent location is provided a file with the name will be created, tested for file locking, and then removed. Providing an existing file will result in an error.

Details

`h5testFileLocking` will create a temporary file and then attempt to apply a file lock using the appropriate function within the HDF5 library. The success or failure of the locking is then recorded and the temporary file removed. Even relatively low level functions such as `H5Fcreate()` will fail inelegantly if file locking fails.

`h5disableFileLocking` will set the environment variable `HDF5_USE_FILE_LOCKING=FALSE`, which is the recommended was to disable this behaviour if file locking is not supported. This will only persist within the current R session. You can set the environment variable outside of R if this is a more general issue on your system.

`h5enableFileLocking` will unset the `HDF5_USE_FILE_LOCKING` environment variable.

More discussion of HDF5's use of file locking can be found online e.g. `https://forum.hdfgroup.org/t/hdf5-1-10-0-and-flock/3761/4` or `https://forum.hdfgroup.org/t/hdf5-files-on-nfs/3985/5`

Value

`h5testFileLocking` returns `TRUE` if a file can be successfully locked at the specified location, or `FALSE` otherwise.

`h5disableFileLocking` and `h5enableFileLocking` set are called for the side effect of setting or unsetting the environment variable `HDF5_USE_FILE_LOCKING` and do not return anything.

Author(s)

Mike Smith

Examples

```
## either a file name or directory can be tested
file <- tempfile()
dir <- tempdir()

h5testFileLocking(dir)
h5testFileLocking(file)

## we can check for file locking, and disable if needed
if (!h5testFileLocking(dir)) {
  h5disableFileLocking()
}
```

`h5_read` *Reads and write object in HDF5 files*

Description

Reads objects in HDF5 files. This function can be used to read either full arrays/vectors or subarrays (hyperslabs) from an existing dataset.

Usage

```
h5read(
  file,
  name,
  index = NULL,
  start = NULL,
  stride = NULL,
  block = NULL,
  count = NULL,
  compoundAsDataFrame = TRUE,
  callGeneric = TRUE,
  read.attributes = FALSE,
  drop = FALSE,
  ...,
  native = FALSE,
  s3 = FALSE,
  s3credentials = NULL
)
```

Arguments

`file`	The file name (character) of the file in which the dataset is be located. It is possible to provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`name`	The name of the dataset in the HDF5 file. The datasets present in `file` can be listed with the function `h5ls`.
`index`	List of indices for subsetting. The length of the list has to agree with the dimensional extension of the HDF5 array. Each list element is an integer vector of indices. A list element equal to NULL chooses all indices in this dimension. Counting is R-style 1-based.
`start`	The start coordinate of a hyperslab (similar to subsetting in R). Counting is R-style 1-based. This argument is ignored, if index is not NULL.
`stride`	The stride of the hypercube. Read the introduction `http://ftp.hdfgroup.org/HDF5/Tutor/phypecont.html` before using this argument. R behaves like Fortran in this example. This argument is ignored, if index is not NULL.
`block`	The block size of the hyperslab. Read the introduction `http://ftp.hdfgroup.org/HDF5/Tutor/phypecont.html` before using this argument. R behaves like Fortran in this example. This argument is ignored, if index is not NULL.
`count`	The number of blocks to be read. This argument is ignored, if index is not NULL.
`compoundAsDataFrame`	If true, a compound datatype will be coerced to a data.frame. This is not possible, if the dataset is multi-dimensional. Otherwise the compound datatype will be returned as a list. Nested compound data types will be returned as a nested list.

`callGeneric`	If TRUE a generic function h5read.classname will be called if it exists depending on the dataset's class attribute within the HDF5 file. This function can be used to convert the standard output of h5read depending on the class attribute. Note that h5read is not a S3 generic function. Dispatching is done based on the HDF5 attribute after the standard h5read function.
`read.attributes`	(logical) If `TRUE`, the HDF5 attributes are read and attached to the respective R object.
`drop`	(logical) If TRUE, the HDF5 object is read as a vector with `NULL` dim attributes.
`...`	Further arguments passed to `H5Dread()`.
`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`
`s3`	Logical value indicating whether the file argument should be treated as a URL to an Amazon S3 bucket, rather than a local file path.
`s3credentials`	A list of length three, providing the credentials for accessing files in a private Amazon S3 bucket.

Details

Read an R object from an HDF5 file. If none of the arguments `start`, `stride`, `block`, `count` are specified, the dataset has the same dimension in the HDF5 file and in memory. If the dataset already exists in the HDF5 file, one can read subarrays, so called hyperslabs from the HDF5 file. The arguments `start`, `stride`, `block`, `count` define the subset of the dataset in the HDF5 file that is to be read/written. See these introductions to hyperslabs: `https://support.hdfgroup.org/HDF5/Tutor/selectsimple.html`, `https://support.hdfgroup.org/HDF5/Tutor/select.html` and `http://ftp.hdfgroup.org/HDF5/Tutor/phypecont.html`. Please note that in R the first dimension is the fastest changing dimension.

When viewing the HDF5 datasets with any C-program (e.g. HDFView), the order of dimensions is inverted. In the R interface counting starts with 1, whereas in the C-programs (e.g. HDFView) counting starts with 0.

Special cases. There are a few instances where rhdf5 will make assumptions about the dataset you are reading and treat it slightly differently. 1) complex numbers. If your datasets is a compound datatype, has only two columns, and these are named 'r' and 'i' rhdf5 will assume the data is intended to be complex numbers and will read this into R's complex type. If that is not the case, you will need to extract the two values separately using the `Re()` and `Im()` accessors manually.

Value

`h5read` returns an array with the data read.

Author(s)

Bernd Fischer, Mike Smith

See Also

h5ls()

Examples

```
h5File <- tempfile(pattern = "ex_hdf5file.h5")
h5createFile(h5File)

# write a matrix
B <- array(seq(0.1, 2.0, by = 0.1), dim = c(5, 2, 2))
h5write(B, h5File, "B")

# read a matrix
E <- h5read(h5File, "B")

# write and read submatrix
h5createDataset(h5File, "S", c(5, 8), storage.mode = "integer", chunk = c(5, 1), level = 7)
h5write(matrix(1:5, nr = 5, nc = 1), file = h5File, name = "S", index = list(NULL, 1))
h5read(h5File, "S")
h5read(h5File, "S", index = list(NULL, 2:3))

# Read a subset of an hdf5 file in a public S3 bucket

h5read("https://rhdf5-public.s3.eu-central-1.amazonaws.com/rhdf5ex_t_float_3d.h5",
  s3 = TRUE, name = "a1", index = list(NULL, 3, NULL)
)
```

---

h5_readAttributes *Read all attributes from a given location in an HDF5 file*

Description

Read all attributes from a given location in an HDF5 file

Usage

```
h5readAttributes(file, name, native = FALSE, ...)
```

Arguments

file	Character vector of length 1, giving the path to the HDF5
name	Path within the HDF5 file to the object whose attributes should be read. The datasets present in `file` can be listed with the function h5ls().
native	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation.
...	Further arguments passed to H5Aread().

Value

A named list of the same length as the number of attributes attached to the specific object. The names of the list entries correspond to the attribute names. If no attributes are found an empty list is returned.

`h5_save` *Saves a one or more objects to an HDF5 file.*

Description

Saves a number of R objects to an HDF5 file.

Usage

```
h5save(..., file, name = NULL, createnewfile = TRUE, native = FALSE)
```

Arguments

`...`	The objects to be saved.
`file`	The filename (character) of the file in which the dataset will be located. It is also possible to provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`name`	A character vector of names for the datasets. The length of the name vector should match the number of objects.
`createnewfile`	If `TRUE`, a new file will be created if necessary.
`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

Details

The objects will be saved to the HDF5 file. If the file does not exists it will be created. The data can be read again by either `h5dump()` or individually for each dataset by `h5read()`.

Value

Nothing returned.

Author(s)

Bernd Fischer

See Also

`h5ls()`, `h5write()`

Examples

```
A <- 1:7
B <- 1:18
D <- seq(0, 1, by = 0.1)

h5File <- tempfile(pattern = "ex_save.h5")
h5save(A, B, D, file = h5File)
h5dump(h5File)
```

`h5_set_extent` *Set a new dataset extension*

Description

Set a new dataset extension to an existing dataset in an HDF5 file

Usage

```
h5set_extent(file, dataset, dims, native = FALSE)
```

Arguments

`file`	The filename (character) of the file in which the dataset will be located. For advanced programmers it is possible to provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`dataset`	The name of the dataset in the HDF5 file, or an object of class H5IdComponent representing a H5 dataset identifier. See `H5Dcreate()`, or `H5Dopen()` to create an object of this kind.
`dims`	The dimensions of the array as they will appear in the file. Note, the dimensions will appear in inverted order when viewing the file with a C program (e.g. HDFView), because the fastest changing dimension in R is the first one, whereas the fastest changing dimension in C is the last one.
`native`	An object of class `logical`. If `TRUE`, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`

Value

Returns `TRUE` if the dimension of the dataset was changed successfully and `FALSE` otherwise.

Author(s)

Bernd Fischer, Mike Smith

Examples

```
tmpfile <- tempfile()
h5createFile(file = tmpfile)
h5createDataset(tmpfile, "A", c(10, 12), c(20, 24))
h5ls(tmpfile, all = TRUE)[c("dim", "maxdim")]
h5set_extent(tmpfile, "A", c(20, 24))
h5ls(tmpfile, all = TRUE)[c("dim", "maxdim")]
```

`h5_write` *Write object to an HDF5 file.*

Description

Writes an R object to an HDF5 file. This function can be used to write either full arrays/vectors or subarrays (hyperslabs) within an existing dataset.

Usage

```
h5write(obj, file, name, ...)

## Default S3 method:
h5write(
  obj,
  file,
  name,
  createnewfile = TRUE,
  write.attributes = FALSE,
  ...,
  native = FALSE
)

h5writeDataset(obj, h5loc, name, ...)

## S3 method for class 'data.frame'
h5writeDataset(
  obj,
  h5loc,
  name,
  level = 6,
  chunk,
  DataFrameAsCompound = TRUE,
  ...
)

## S3 method for class 'array'
h5writeDataset(
  obj,
  h5loc,
  name,
  index = NULL,
  start = NULL,
  stride = NULL,
  block = NULL,
  count = NULL,
  size = NULL,
  variableLengthString = FALSE,
  encoding = NULL,
  level = 6,
  ...
)
```

Arguments

`obj`	The R object to be written.
`file`	The filename (character) of the file in which the dataset will be located. For advanced programmers it is possible to provide an object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`name`	The name of the dataset in the HDF5 file.
`...`	Further arguments passed to `H5Dwrite()`.
`createnewfile`	If `TRUE`, a new file will be created if necessary.
`write.attributes`	(logical) If TRUE, all R-attributes attached to the object `obj` are written to the HDF5 file.
`native`	An object of class `logical`. If TRUE, array-like objects are treated as stored in HDF5 row-major rather than R column-major orientation. Using `native = TRUE` increases HDF5 file portability between programming languages. A file written with `native = TRUE` should also be read with `native = TRUE`
`h5loc`	An object of class H5IdComponent representing a H5 location identifier (file or group). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()` to create an object of this kind.
`level`	The compression level. An integer value between 0 (no compression) and 9 (highest and slowest compression). Only used, if the dataset does not yet exist. See `h5createDataset()` to create an dataset.
`chunk`	Specifies the number of items to be include in an HDF5 chunk. If left unspecified the defaults is the smaller of: the total number of cols or the number of cols that fit within 4GB of memory. If `DataFrameAsCompound=FALSE` each row of the `data.frame` can be consider an "col".
`DataFrameAsCompound`	If true, a `data.frame` will be saved as a compound data type. Otherwise it is saved like a list. The advantage of saving a data.frame as a compound data type is that it can be read as a table from python or with a struct-type from C. The

disadvantage is that the data has to be rearranged on disk and thus can slow down I/O. If fast reading is required, `DataFrameAsCompound=FALSE` is recommended.

`index`	List of indices for subsetting. The length of the list has to agree with the dimensional extension of the HDF5 array. Each list col is an integer vector of indices. A list col equal to `NULL` chooses all indices in this dimension. Counting is R-style 1-based.
`start`	The start coordinate of a hyperslab (similar to subsetting in R). Counting is R-style 1-based. This argument is ignored, if index is not NULL.
`stride`	The stride of the hypercube. Read the introduction `http://ftp.hdfgroup.org/HDF5/Tutor/phypecont.html` before using this argument. R behaves like Fortran in this example. This argument is ignored, if index is not NULL.
`block`	The block size of the hyperslab. Read the introduction `http://ftp.hdfgroup.org/HDF5/Tutor/phypecont.html` before using this argument. R behaves like Fortran in this example. This argument is ignored, if index is not NULL.
`count`	The number of blocks to be written. This argument is ignored, if index is not NULL.
`size`	The length of the fixed-width string data type, when `obj` is a character vector. If `NULL`, this is set to the length of the largest string.
`variableLengthString`	Whether character vectors should be written as variable-length strings into the attributes. If `TRUE`, `size` is ignored.
`encoding`	The encoding of the string data type. Valid options are "ASCII" or "UTF-8".

Details

Writes an R object to an HDF5 file. If none of the arguments `start`, `stride`, `block`, `count` is specified, the dataset has the same dimension in the HDF5 file and in memory. If the dataset already exists in the HDF5 file, one can write subarrays, (so called hyperslabs) to the HDF5 file. The arguments `start`, `stride`, `block`, `count` define the subset of the dataset in the HDF5 file that is to be written to. See these introductions to hyperslabs: `https://support.hdfgroup.org/HDF5/Tutor/selectsimple.html`, `https://support.hdfgroup.org/HDF5/Tutor/select.html` and `http://ftp.hdfgroup.org/HDF5/Tutor/phypecont.html`. Please note that in R the first dimension is the fastest changing dimension.

When viewing the HDF5 datasets with any C-program (e.g. HDFView), the order of dimensions is inverted. In the R interface counting starts with 1, whereas in the C-programs (e.g. HDFView) counting starts with 0.

If code `obj` is of type ’complex’ then it will be written as a compound datatype to the HDF5, with cols named ’r’ and ’i’ for the real and imaginary parts respectively.

Value

`h5write` returns 0 if successful.

Author(s)

Bernd Fischer, Mike Smith

References

https://portal.hdfgroup.org/display/HDF5

See Also

`h5ls()`, `h5createFile()`, `h5createDataset()`, rhdf5

Examples

```
h5File <- tempfile(fileext = ".h5")
h5createFile(h5File)

# write a matrix
B <- array(seq(0.1, 2.0, by = 0.1), dim = c(5, 2, 2))
attr(B, "scale") <- "liter"
h5write(B, h5File, "B")

# write a submatrix
h5createDataset(h5File, "S", c(5, 8), storage.mode = "integer", chunk = c(5, 1), level = 7)
h5write(matrix(1:5, nr = 5, nc = 1), file = h5File, name = "S", index = list(NULL, 1))
```

`h5_writeAttribute` *Write an R object as an HDF5 attribute*

Description

Write an R object as an HDF5 attribute

Usage

```
h5writeAttribute(
  attr,
  h5obj,
  name,
  h5loc,
  encoding = NULL,
  variableLengthString = FALSE,
  asScalar = FALSE,
  checkForNA = TRUE
)

## S3 method for class 'array'
h5writeAttribute(
  attr,
  h5obj,
  name,
  h5loc,
  encoding = NULL,
  variableLengthString = FALSE,
  asScalar = FALSE,
  checkForNA = TRUE
)
```

Arguments

`attr`	The R object to be written as an HDF5 attribute.
`h5obj`	Normally an object of class H5IdComponent representing a H5 object identifier (file, group, or dataset). See `H5Fcreate()`, `H5Fopen()`, `H5Gcreate()`, `H5Gopen()`, `H5Dcreate()`, or `H5Dopen()` to create an object of this kind. This argument can also be given the path to an HDF5 file.
`name`	The name of the attribute to be written.
`h5loc`	The location of the group or dataset within a file to which the attribute should be attached. This argument is only used if the `h5obj` argument is the path to an HDF5 file, otherwise it is ignored.
`encoding`	The encoding of the string data type. Valid options are "ASCII" and "UTF-8".
`variableLengthString`	Whether character vectors should be written as variable-length strings into the attributes.
`asScalar`	Whether length-1 `attr` should be written into a scalar dataspace.
`checkForNA`	Whether a `attr` should be checked for `NA` values before being written. This only applies of `attr` is of type logical. Testing for `NA` values can be slow if the object to be written is large, so if you are sure no such values will be present this argument can be used to disable the testing.

`rhdf5` *rhdf5: An interface between HDF5 and R*

Description

The rhdf5 package provides two categories of functions:

- `h5` functions are high-level R functions that provide a convenient way of accessing HDF5 files
- `H5` functions mirror much of the the HDF5 C API

Index