Package ‘h5lite’

January 29, 2026

Title Simplified 'HDF5' Interface

Version 2.0.0.2

Description A user-friendly interface for the Hierarchical Data Format 5 ('HDF5') library designed to ``just work.'' It bundles the necessary system libraries to ensure easy installation on all platforms. Features smart defaults that automatically map R objects (vectors, matrices, data frames) to efficient 'HDF5' types, removing the need to manage low-level details like dataspaces or property lists. Uses the 'HDF5' library developed by The HDF Group <https://www.hdfgroup.org/>.

URL https://github.com/cmmr/h5lite, https://cmmr.github.io/h5lite/

BugReports https://github.com/cmmr/h5lite/issues

Depends R (>= 4.2.0)

LinkingTo hdf5lib (>= 2.0.0.5)

Suggests bit64, knitr, rmarkdown, testthat (>= 3.0.0)

NeedsCompilation yes

License MIT + file LICENSE

Encoding UTF-8

Config/testthat/edition 3

Config/Needs/website rmarkdown

RoxygenNote 7.3.3

VignetteBuilder knitr

Author Daniel P. Smith [aut, cre] (ORCID: <https://orcid.org/0000-0002-2479-2044>), Alkek Center for Metagenomics and Microbiome Research [cph, fnd]

Maintainer Daniel P. Smith <dansmith01@gmail.com>

Repository CRAN

Date/Publication 2026-01-29 20:20:07 UTC

Contents

`h5_attr_names` *List HDF5 Attributes*

Description

Lists the names of attributes attached to a specific HDF5 object.

Usage

```
h5_attr_names(file, name = "/")
```

Arguments

`file`	The path to the HDF5 file.
`name`	The path to the object (dataset or group) to query. Use `/` for the file's root attributes.

Value

A character vector of attribute names.

See Also

`h5_ls()`

Examples

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

h5_write(1:10,          file, "data")
h5_write(I("meters"),   file, "data", attr = "unit")
h5_write(I(Sys.time()), file, "data", attr = "timestamp")

h5_attr_names(file, "data") # "unit" "timestamp"

unlink(file)
```

---

h5_class *Get R Class of an HDF5 Object or Attribute*

Description

Inspects an HDF5 object (or an attribute attached to it) and returns the R class that `h5_read()` would produce.

Usage

```
h5_class(file, name, attr = NULL)
```

Arguments

`file`	The path to the HDF5 file.
`name`	The full path of the object (group or dataset) to check.
`attr`	The name of an attribute to check. If `NULL` (default), the function checks the class of the object itself.

Details

This function determines the resulting R class by inspecting the storage metadata.

- **Group** $\rightarrow$ `"list"`
- **Integer** $\rightarrow$ `"numeric"`
- **Floating Point** $\rightarrow$ `"numeric"`
- **String** $\rightarrow$ `"character"`
- **Complex** $\rightarrow$ `"complex"`
- **Enum** $\rightarrow$ `"factor"`
- **Opaque** $\rightarrow$ `"raw"`
- **Compound** $\rightarrow$ `"data.frame"`
- **Null** $\rightarrow$ `"NULL"`

Value

A character string representing the R class (e.g., "integer", "numeric", "complex", "character", "factor", "raw", "list", "NULL"). Returns NA_character_ for HDF5 types that h5lite cannot read.

See Also

h5_typeof(), h5_read()

Examples

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

h5_write(1:10, file, "my_ints", as = "int32")
h5_class(file, "my_ints") # "numeric"

h5_write(mtcars, file, "mtcars")
h5_class(file, "mtcars") # "data.frame"

h5_write(c("a", "b", "c"), file, "strings")
h5_class(file, "strings") # "character"

h5_write(c(1, 2, 3), file, "my_floats", as = "float64")
h5_class(file, "my_floats") # "numeric"

unlink(file)
```

---

h5_create_file *Create an HDF5 File*

Description

Explicitly creates a new, empty HDF5 file.

Usage

```
h5_create_file(file)
```

Arguments

file	Path to the HDF5 file to be created.

Details

This function is a simple wrapper around h5_create_group(file, "/"). Its main purpose is to allow for explicit file creation in code.

Note that calling this function is almost always **unnecessary**, as all h5lite writing functions (like h5_write() or h5_create_group()) will automatically create the file if it does not exist.

It is provided as a convenience for users who prefer to explicitly create a file before writing data to it.

Value

Invisibly returns `NULL`. This function is called for its side effects.

File Handling

- If `file` does not exist, it will be created as a new, empty HDF5 file.
- If `file` already exists and is a valid HDF5 file, this function does nothing and returns successfully.
- If `file` exists but is **not** a valid HDF5 file (e.g., a text file), an error will be thrown and the file will not be modified.

See Also

`h5_create_group()`, `h5_write()`

Examples

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

# Explicitly create the file
h5_create_file(file)

if (file.exists(file)) {
  message("File created successfully.")
}

unlink(file)
```

`h5_create_group` *Create an HDF5 Group*

Description

Explicitly creates a new group (or nested groups) in an HDF5 file. This is useful for creating an empty group structure.

Usage

```
h5_create_group(file, name)
```

Arguments

`file`	The path to the HDF5 file.
`name`	The full path of the group to create (e.g., "/g1/g2").

Value

Invisibly returns `NULL`. This function is called for its side effects.

Examples

```
file <- tempfile(fileext = ".h5")
h5_create_file(file)

# Create a nested group structure
h5_create_group(file, "/data/experiment/run1")
h5_ls(file)

unlink(file)
```

---

h5_delete *Delete an HDF5 Object or Attribute*

Description

Deletes an object (dataset or group) or an attribute from an HDF5 file. If the object or attribute does not exist, a warning is issued and the function returns successfully (no error is raised).

Usage

```
h5_delete(file, name, attr = NULL, warn = TRUE)
```

Arguments

`file`	The path to the HDF5 file.
`name`	The full path of the object to delete (e.g., `"/data/dset"` or `"/groups/g1"`).
`attr`	The name of the attribute to delete. • If `NULL` (the default), the object specified by `name` is deleted. • If a string is provided, the attribute named `attr` is removed from the object `name`.
`warn`	Emit a warning if the name/attr does not exist. Default: `TRUE`

Value

Invisibly returns `NULL`. This function is called for its side effects.

See Also

`h5_create_group()`, `h5_move()`

Examples

```
file <- tempfile(fileext = ".h5")
h5_create_file(file)

# Create some data and attributes
h5_write(matrix(1:10, 2, 5), file, "matrix")
h5_write("A note", file, "matrix", attr = "note")

# Review the file structure
h5_str(file)

# Delete the attribute
h5_delete(file, "matrix", attr = "note")

# Review the file structure
h5_str(file)

# Delete the dataset
h5_delete(file, "matrix")

# Review the file structure
h5_str(file)

# Cleaning up
unlink(file)
```

`h5_dim` *Get Dimensions of an HDF5 Object or Attribute*

Description

Returns the dimensions of a dataset or an attribute as an integer vector. These dimensions match the R-style (column-major) interpretation.

Usage

```
h5_dim(file, name, attr = NULL)
```

Arguments

`file`	The path to the HDF5 file.
`name`	Name of the dataset or object.
`attr`	The name of an attribute to check. If `NULL` (default), the function returns the dimensions of the object itself.

Value

An integer vector of dimensions, or `integer(0)` for scalars.

Examples

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

h5_write(matrix(1:10, 2, 5), file, "matrix")
h5_dim(file, "matrix") # 2 5

h5_write(mtcars, file, "mtcars")
h5_dim(file, "mtcars") # 32 11

h5_write(I(TRUE), file, "my_bool")
h5_dim(file, "my_bool") # integer(0)

h5_write(1:10, file, "my_ints")
h5_dim(file, "my_ints") # 10

unlink(file)
```

`h5_exists` *Check if an HDF5 File, Object, or Attribute Exists*

Description

Safely checks if a file, an object within a file, or an attribute on an object exists.

Usage

```
h5_exists(file, name = "/", attr = NULL, assert = FALSE)
```

Arguments

`file`	Path to the file.
`name`	The full path of the object to check (e.g., `"/data/matrix"`). Defaults to `"/"` to test file validity.
`attr`	The name of an attribute to check. If provided, the function tests for the existence of this attribute on `name`.
`assert`	Logical. If `TRUE` and the target does not exist, the function will stop with an informative error message instead of returning `FALSE`. Defaults to `FALSE`.

Details

This function provides a robust, error-free way to test for existence.

- **Testing for a File:** If `name` is `/` and `attr` is `NULL`, the function checks if `file` is a valid, readable HDF5 file.
- **Testing for an Object:** If `name` is a path (e.g., `/data/matrix`) and `attr` is `NULL`, the function checks if the specific object exists.
- **Testing for an Attribute:** If `attr` is provided, the function checks if that attribute exists on the specified object `name`.

Value

A logical value: `TRUE` if the target exists and is valid, `FALSE` otherwise.

See Also

`h5_is_group()`, `h5_is_dataset()`

Examples

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

h5_exists(file) # FALSE

h5_create_file(file)
h5_exists(file) # TRUE

h5_exists(file, "missing_object") # FALSE

h5_write(1:10, file, "my_ints")
h5_exists(file, "my_ints") # TRUE

h5_exists(file, "my_ints", "missing_attr") # FALSE

h5_write(1:10, file, "my_ints", attr = "my_attr")
h5_exists(file, "my_ints", "my_attr") # TRUE

unlink(file)
```

`h5_is_dataset` *Check if an HDF5 Object is a Dataset*

Description

Checks if the object at a given path is a dataset.

Usage

```
h5_is_dataset(file, name, attr = NULL)
```

Arguments

`file`	The path to the HDF5 file.
`name`	The full path of the object to check.
`attr`	The name of an attribute. If provided, the function returns `TRUE` if the attribute exists, as all attributes are considered datasets in HDF5 context. (Default: `NULL`)

Value

A logical value: TRUE if the object exists and is a dataset, FALSE otherwise (if it is a group, or does not exist).

See Also

h5_is_group(), h5_exists()

Examples

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

h5_write(1, file, "dset")
h5_is_dataset(file, "dset") # TRUE

h5_create_group(file, "grp")
h5_is_dataset(file, "grp") # FALSE

h5_write(1, file, "grp", attr = "my_attr")
h5_is_dataset(file, "grp", "my_attr") # TRUE

unlink(file)
```

---

h5_is_group *Check if an HDF5 Object is a Group*

Description

Checks if the object at a given path is a group.

Usage

```
h5_is_group(file, name, attr = NULL)
```

Arguments

file	The path to the HDF5 file.
name	The full path of the object to check.
attr	The name of an attribute. This parameter is included for consistency with other functions. Since attributes cannot be groups, providing this will always return FALSE. (Default: NULL)

Value

A logical value: TRUE if the object exists and is a group, FALSE otherwise (if it is a dataset, or does not exist).

See Also

`h5_is_dataset()`, `h5_exists()`

Examples

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

h5_create_group(file, "grp")
h5_is_group(file, "grp") # TRUE

h5_write(1:10, file, "my_ints")
h5_is_group(file, "my_ints") # FALSE

unlink(file)
```

`h5_length` *Get the Total Length of an HDF5 Object or Attribute*

Description

Behaves like `length()` for R objects.

- For **Compound Datasets** (data.frames), this is the number of columns.
- For **Datasets** and **Attributes**, this is the product of all dimensions (total number of elements).
- For **Groups**, this is the number of objects directly contained in the group.
- Scalar datasets or attributes return 1.

Usage

```
h5_length(file, name, attr = NULL)
```

Arguments

`file`	The path to the HDF5 file.
`name`	The full path of the object (group or dataset).
`attr`	The name of an attribute to check. If provided, the length of the attribute is returned.

Value

An integer representing the total length (number of elements).

Examples

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

h5_write(1:100, file, "my_vec")
h5_length(file, "my_vec") # 100

h5_write(mtcars, file, "my_df")
h5_length(file, "my_df") # 11 (ncol(mtcars))

h5_write(as.matrix(mtcars), file, "my_mtx")
h5_length(file, "my_mtx") # 352 (prod(dim(mtcars)))

h5_length(file, "/") # 3

unlink(file)
```

`h5_ls` *List HDF5 Objects*

Description

Lists the names of objects (datasets and groups) within an HDF5 file or group.

Usage

```
h5_ls(file, name = "/", recursive = TRUE, full.names = FALSE, scales = FALSE)
```

Arguments

`file`	The path to the HDF5 file.
`name`	The group path to start listing from. Defaults to the root group (`/`).
`recursive`	If `TRUE` (default), lists all objects found recursively under `name`. If `FALSE`, lists only the immediate children.
`full.names`	If `TRUE`, the full paths from the file's root are returned. If `FALSE` (the default), names are relative to `name`.
`scales`	If `TRUE`, also returns datasets that are dimensions scales for other datasets.

Value

A character vector of object names. If `name` is `/` (the default), the paths are relative to the root of the file. If `name` is another group, the paths are relative to that group (unless `full.names = TRUE`).

See Also

`h5_attr_names()`, `h5_str()`

Examples

```
file <- tempfile(fileext = ".h5")
h5_create_group(file, "foo/bar")
h5_write(1:5, file, "foo/data")

# List everything recursively
h5_ls(file)

# List only top-level objects
h5_ls(file, recursive = FALSE)

# List relative to a sub-group
h5_ls(file, "foo")

unlink(file)
```

`h5_move` *Move or Rename an HDF5 Object*

Description

Moves or renames an object (dataset, group, etc.) within an HDF5 file.

Usage

```
h5_move(file, from, to)
```

Arguments

`file`	The path to the HDF5 file.
`from`	The current (source) path of the object (e.g., `"/group/data"`).
`to`	The new (destination) path for the object (e.g., `"/group/data_new"`).

Details

This function provides an efficient, low-level wrapper for the HDF5 library's `H5Lmove` function. It is a metadata-only operation, meaning the data itself is not read or rewritten. This makes it extremely fast, even for very large datasets.

You can use this function to either rename an object within the same group (e.g., `"data/old"` to `"data/new"`) or to move an object to a different group (e.g., `"data/old"` to `"archive/old"`). The destination parent group will be automatically created if it does not exist.

Value

This function is called for its side-effect and returns `NULL` invisibly.

See Also

`h5_create_group()`, `h5_delete()`

Examples

```
file <- tempfile(fileext = ".h5")
h5_write(1:10, file, "group/dataset")

# Review the file structure
h5_str(file)

# Rename within the same group
h5_move(file, "group/dataset", "group/renamed")

# Review the file structure
h5_str(file)

# Move to a new group (creates parent automatically)
h5_move(file, "group/renamed", "archive/dataset")

# Review the file structure
h5_str(file)

unlink(file)
```

---

h5_names *Get Names of an HDF5 Object*

Description

Returns the names of the object.

- For **Groups**, it returns the names of the objects contained in the group (similar to `ls()`).
- For **Compound Datasets** (data.frames), it returns the column names.
- For other **Datasets**, it looks for a dimension scale and returns it if found.

Usage

```
h5_names(file, name = "/", attr = NULL)
```

Arguments

`file`	The path to the HDF5 file.
`name`	The full path of the object.
`attr`	The name of an attribute. If provided, returns the names associated with the attribute (e.g., field names if the attribute is a compound type). (Default: `NULL`)

Value

A character vector of names, or `NULL` if the object has no names.

Examples

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

h5_write(data.frame(x=1, y=2), file, "df")
h5_names(file, "df") # "x" "y"

x <- 1:5
names(x) <- letters[1:5]
h5_write(x, file, "x")
h5_names(file, "x") # "a" "b" "c" "d" "e"

h5_write(mtcars[,c("mpg", "hp")], file, "dset")
h5_names(file, "dset") # "mpg" "hp"

unlink(file)
```

`h5_open` *Create an HDF5 File Handle*

Description

Creates a file handle that provides a convenient, object-oriented interface for interacting with and navigating a specific HDF5 file.

Usage

```
h5_open(file)
```

Arguments

`file`	Path to the HDF5 file. The file will be created if it does not exist.

Details

This function returns a special `h5` object that wraps the standard `h5lite` functions. The primary benefit is that the `file` argument is pre-filled, allowing for more concise and readable code when performing multiple operations on the same file.

For example, instead of writing:

```
h5_write(1:10, file, "dset1")
h5_write(2:20, file, "dset2")
h5_ls(file)
```

You can create a handle and use its methods. Note that the `file` argument is omitted from the method calls:

```
h5 <- h5_open("my_file.h5")
h5$write(1:10, "dset1")
h5$write(2:20, "dset2")
h5$ls()
h5$close()
```

Value

An object of class `h5` with methods for interacting with the file.

Pass-by-Reference Behavior

Unlike most R objects, the `h5` handle is an **environment**. This means it is passed by reference. If you assign it to another variable (e.g., `h5_alias <- h5`), both variables point to the *same* handle. Modifying one (e.g., by calling `h5_alias$close()`) will also affect the other.

Interacting with the HDF5 File

The `h5` object provides several ways to interact with the HDF5 file:

Standard `h5lite` Functions as Methods: Most `h5lite` functions (e.g., `h5_read`, `h5_write`, `h5_ls`) are available as methods on the `h5` object, without the `h5_` prefix.

For example, `h5$write(data, "dset")` is equivalent to `h5_write(data, file, "dset")`.

The available methods are: `attr_names`, `cd`, `class`, `close`, `create_group`, `delete`, `dim`, `exists`, `is_dataset`, `is_group`, `length`, `ls`, `move`, `names`, `pwd`, `read`, `str`, `typeof`, `write`.

Navigation (`$cd()`, `$pwd()`): The handle maintains an internal working directory to simplify path management.

- `h5$cd(group)`: Changes the handle's internal working directory. This is a stateful, pass-by-reference operation. It understands absolute paths (e.g., `"/new/path"`) and relative navigation (e.g., `"../other"`). The target group does not need to exist.
- `h5$pwd()`: Returns the current working directory.

When you call a method like `h5$read("dset")`, the handle automatically prepends the current working directory to any relative path. If you provide an absolute path (e.g., `h5$read("/path/to/dset")`), the working directory is ignored.

Closing the Handle (`$close()`): The `h5lite` package does not keep files persistently open. Each operation opens, modifies, and closes the file. Therefore, the `h5$close()` method does not perform any action on the HDF5 file itself.

Its purpose is to invalidate the handle, preventing any further operations from being called. After `h5$close()` is called, any subsequent method call (e.g., `h5$ls()`) will throw an error.

Examples

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

# Open the handle
h5 <- h5_open(file)

# Write data (note: 'data' is the first argument, 'file' is implicit)
h5$write(1:5, "vector")
h5$write(matrix(1:9, 3, 3), "matrix")

# Create a group and navigate to it
h5$create_group("simulations")
h5$cd("simulations")
print(h5$pwd()) # "/simulations"

# Write data relative to the current working directory
h5$write(rnorm(10), "run1") # Writes to /simulations/run1

# Read data
dat <- h5$read("run1")

# List contents of current WD
h5$ls()

# Close the handle
h5$close()
unlink(file)
```

h5_read *Read an HDF5 Object or Attribute*

Description

Reads a dataset, a group, or a specific attribute from an HDF5 file into an R object.

Usage

```
h5_read(file, name = "/", attr = NULL, as = "auto")
```

Arguments

`file`	The path to the HDF5 file.
`name`	The full path of the dataset or group to read (e.g., `"/data/matrix"`).
`attr`	The name of an attribute to read. • If `NULL` (default), the function reads the object specified by `name` (and attaches its attributes to the result). • If provided (string), the function reads *only* the specified attribute from `name`.

`as` The target R data type.

- **Global:** `"auto"` (default), `"integer"`, `"double"`, `"logical"`, `"bit64"`, `"null"`.
- **Specific:** A named vector mapping names or type classes to R types (see Section "Type Conversion").

Value

An R object corresponding to the HDF5 object or attribute. Returns `NULL` if the object is skipped via `as = "null"`.

Type Conversion (`as`)

You can control how HDF5 data is converted to R types using the `as` argument.

1. Mapping by Name:

- `as = c("data_col" = "integer")`: Reads the dataset/column named "data_col" as an integer.
- `as = c("@validated" = "logical")`: When reading a dataset, this forces the attached attribute "validated" to be read as logical.

2. Mapping by HDF5 Type Class: You can target specific HDF5 data types using keys prefixed with a dot (`.`). Supported classes include:

- **Integer:** `.int`, `.int8`, `.int16`, `.int32`, `.int64`
- **Unsigned:** `.uint`, `.uint8`, `.uint16`, `.uint32`, `.uint64`
- **Floating Point:** `.float`, `.float16`, `.float32`, `.float64`

Example: `as = c(.uint8 = "logical", .int = "bit64")`

3. Precedence & Attribute Config:

- **Attributes vs Datasets:** Attribute type mappings take precedence over dataset mappings. If you specify `as = c(.uint = "logical", "@.uint" = "integer")`, unsigned integer datasets will be read as `logical`, but unsigned integer *attributes* will be read as `integer`.
- **Specific vs Generic:** Specific keys (e.g., `.uint32`) take precedence over generic keys (e.g., `.uint`), which take precedence over the global default (`.`).

Note

The `@` prefix is **only** used to configure attached attributes when reading a dataset (`attr = NULL`). If you are reading a specific attribute directly (e.g., `h5_read(..., attr = "id")`), do **not** use the `@` prefix in the `as` argument.

See Also

`h5_write()`

Examples

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

# --- Write Data ---
h5_write(c(10L, 20L), file, "ints")
h5_write(I(TRUE),     file, "ints", attr = "ready")
h5_write(c(10.5, 18), file, "floats")
h5_write(I("meters"), file, "floats", attr = "unit")

# --- Read Data ---
# Read dataset
x <- h5_read(file, "ints")
print(x)

# Read dataset with attributes
y <- h5_read(file, "floats")
print(attr(y, "unit"))

# Read a specific attribute directly
unit <- h5_read(file, "floats", attr = "unit")
print(unit)

# --- Type Conversion Examples ---

# Force integer dataset to be read as numeric (double)
x_dbl <- h5_read(file, "ints", as = "double")
class(x_dbl)

# Force attached attribute to be read as logical
# Note the "@" prefix to target the attribute
z <- h5_read(file, "ints", as = c("@ready" = "logical"))
print(z)

unlink(file)
```

`h5_str` *Display the Structure of an HDF5 Object*

Description

Recursively prints a summary of an HDF5 group or dataset, similar to the structure of `h5ls -r`. It displays the nested structure, object types, dimensions, and attributes.

Usage

```
h5_str(file, name = "/", attrs = TRUE, members = TRUE, markup = interactive())
```

Arguments

`file`	The path to the HDF5 file.
`name`	The name of the group or dataset to display. Defaults to the root group "/".
`attrs`	Set to `FALSE` to hide attributes. The default (`TRUE`) shows attributes prefixed with `@`.
`members`	Set to `FALSE` to hide compound dataset members. The default (`TRUE`) shows members prefixed with `$`.
`markup`	Set to `FALSE` to remove colors and italics from the output.

Details

This function provides a quick and convenient way to inspect the contents of an HDF5 file. It performs a recursive traversal of the file from the C-level and prints a formatted summary to the R console.

This function **does not read any data** into R. It only inspects the metadata (names, types, dimensions) of the objects in the file, making it fast and memory-safe for arbitrarily large files.

Value

This function is called for its side-effect of printing to the console and returns `NULL` invisibly.

See Also

`h5_ls()`, `h5_attr_names()`

Examples

```
file <- tempfile(fileext = ".h5")
h5_write(list(x = 1:10, y = matrix(1:9, 3, 3)), file, "group")
h5_write("metadata", file, "group", attr = "info")

# Print structure
h5_str(file)

unlink(file)
```

`h5_typeof` *Get HDF5 Storage Type of an Object or Attribute*

Description

Returns the low-level HDF5 storage type of a dataset or an attribute (e.g., "int8", "float64", "utf8", "ascii[10]"). This allows inspecting the file storage type before reading the data into R.

Usage

```
h5_typeof(file, name, attr = NULL)
```

Arguments

`file`	The path to the HDF5 file.
`name`	Name of the dataset or object.
`attr`	The name of an attribute to check. If `NULL` (default), the function returns the type of the object itself.

Value

A character string representing the HDF5 storage type (e.g., "float32", "uint32", "ascii[10]", "compound[2]").

See Also

`h5_class()`, `h5_exists()`

Examples

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

h5_write(1L, file, "int32_val", as = "int32")
h5_typeof(file, "int32_val") # "int32"

h5_write(mtcars, file, "mtcars")
h5_typeof(file, "mtcars") # "compound[11]"

h5_write(c("a", "b", "c"), file, "strings")
h5_typeof(file, "strings") # "utf8[1]"

unlink(file)
```

`h5_write` *Write an R Object to HDF5*

Description

Writes an R object to an HDF5 file, creating the file if it does not exist. This function acts as a unified writer for datasets, groups (lists), and attributes.

Usage

```
h5_write(data, file, name, attr = NULL, as = "auto", compress = TRUE)
```

Arguments

`data`	The R object to write. Supported: `numeric`, `complex`, `logical`, `character`, `factor`, `raw`, `matrix`, `data.frame`, `integer64`, `POSIXt`, `NULL`, and nested `lists`.
`file`	The path to the HDF5 file.
`name`	The name of the dataset or group to write (e.g., "/data/matrix").
`attr`	The name of an attribute to write. • If `NULL` (default), `data` is written as a dataset or group at the path `name`. • If provided (string), `data` is written as an attribute named `attr` attached to the object `name`.
`as`	The target HDF5 data type. Defaults to `"auto"`. See the **Data Type Selection** section for a full list of valid options (including `"int64"`, `"bfloat16"`, `"utf8[n]"`, etc.) and how to map sub-components of `data`.
`compress`	Compression configuration. • `TRUE` (default): Enables compression (zlib level 5). • `FALSE` or `0`: Disables compression. • Integer `1-9`: Specifies the zlib compression level.

Value

Invisibly returns `file`. This function is called for its side effects.

Writing Scalars

By default, `h5_write` saves single-element vectors as 1-dimensional arrays. To write a true HDF5 scalar, wrap the value in `I()` to treat it "as-is."

Examples:

```
h5_write(I(5), file, "x") # Creates a scalar dataset
h5_write(5, file, "x")    # Creates a 1D array of length 1
```

Data Type Selection (`as` Argument)

By default, `as = "auto"` will automatically select the most appropriate data type for the given object. For numeric types, this will be the smallest type that can represent all values in the vector. For character types, `h5lite` will use a ragged vs rectangular heuristic, favoring small file size over fast I/O. For R data types not mentioned below, see `vignette("data-types")` for information on their fixed mappings to HDF5 data types.

Numeric and Logical Vectors:
When writing a numeric or logical vector, you can specify one of the following storage types for it:

- **Floating Point:** `"float16"`, `"float32"`, `"float64"`, `"bfloat16"`
- **Signed Integer:** `"int8"`, `"int16"`, `"int32"`, `"int64"`
- **Unsigned Integer:** `"uint8"`, `"uint16"`, `"uint32"`, `"uint64"`

NOTE: `NA` values must be stored as `float64`. `NaN`, `Inf`, and `-Inf` must be stored as a floating point type.

Examples:

```
h5_write(1:100, file, "big_ints", as = "int64")
h5_write(TRUE,  file, "my_bool",  as = "float32")
```

Character Vectors:

You can control whether character vectors are stored as variable or fixed length strings, and whether to use UTF-8 or ASCII encoding.

- **Variable Length Strings:** `"utf8"`, `"ascii"`
- **Fixed Length Strings:**
  - `"utf8[]"` or `"ascii[]"` (length is set to the longest string)
  - `"utf8[n]"` or `"ascii[n]"` (where `n` is the length in bytes)

NOTE: Variable-length strings allow for `NA` values but cannot be compressed on disk. Fixed-length strings allow for compression but do not support `NA`.

Examples:

```
h5_write(letters[1:5],    file, "len10_strs", as = "utf8[10]")
h5_write(c('X', 'Y', NA), file, "var_chars",  as = "ascii")
```

Lists, Data Frames, and Attributes:

Provide a named vector to apply type mappings to sub-components of `data`. Set `"skip"` as the type to skip a specific component.

- **Specific Name:** `"col_name" = "type"` (e.g., `c(score = "float32")`)
- **Specific Attribute:** `"@attr_name" = "type"`
- **Class-based:** `".integer" = "type"`, `".numeric" = "type"`
- **Class-based Attribute:** `"@.character" = "type"`, `"@.logical" = "type"`
- **Global Fallback:** `"." = "type"`
- **Global Attribute Fallback:** `"@." = "type"`

Examples:

```
# To strip attributes when writing:
h5_write(data, file, 'no_attrs_obj', as = c('@.' = "skip"))

# To only save the `hp` and `wt` columns:
h5_write(mtcars, file, 'my_df', as = c('hp' = "auto", 'wt' = "float32", '.' = "skip"))
```

Dimension Scales

`h5lite` automatically writes `names`, `row.names`, and `dimnames` as HDF5 dimension scales. Named vectors will generate an `<name>_names` dataset. A data.frame with row names will generate an `<name>_rownames` dataset (column names are saved internally in the original dataset). Matrices will generate `<name>_rownames` and `<name>_colnames` datasets. Arrays will generate `<name>_dimscale_1`, `<name>_dimscale_2`, etc. Special HDF5 metadata attributes link the dimension scales to the dataset. The dimension scales can be relocated with `h5_move()` without breaking the link.

See Also

h5_read()

Examples

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

# 1. Writing Basic Datasets
h5_write(1:10, file, "data/integers")
h5_write(rnorm(10), file, "data/floats")
h5_write(letters[1:5], file, "data/chars")

# 2. Writing Attributes
# Write an object first
h5_write(1:10, file, "data/vector")
# Attach an attribute to it using the 'attr' parameter
h5_write(I("My Description"), file, "data/vector", attr = "description")
h5_write(I(100), file, "data/vector", attr = "scale_factor")

# 3. Controlling Data Types
# Store values as 32-bit signed integers
h5_write(1:5, file, "small_ints", as = "int32")

# 4. Writing Complex Structures (Lists/Groups)
my_list <- list(
  meta    = list(id = 1, name = "Experiment A"),
  results = matrix(runif(9), 3, 3),
  valid   = I(TRUE)
)
h5_write(my_list, file, "experiment_1", as = c(id = "uint16"))

# 5. Writing Data Frames (Compound Datasets)
df <- data.frame(
  id    = 1:5,
  score = c(10.5, 9.2, 8.4, 7.1, 6.0),
  grade = factor(c("A", "A", "B", "C", "D"))
)
h5_write(df, file, "records/scores", as = c(grade = "ascii[1]"))

# 6. Fixed-Length Strings
h5_write(c("A", "B"), file, "fixed_str", as = "ascii[10]")

# 7. Review the file structure
h5_str(file)

# 8. Clean up
unlink(file)
```

Index