Package ‘HDF5Array’

October 16, 2018

Title HDF5 back end for DelayedArray objects

Description An array-like container for convenient access and manipulation of HDF5 datasets. Supports delayed operations and block processing.

Version 1.8.1

Encoding UTF-8

Author Hervé Pagès

Maintainer Hervé Pagès <hpages@fredhutch.org>

biocViews Infrastructure, DataRepresentation, Sequencing, Annotation, Coverage, GenomeAnnotation

Depends R (>= 3.4), methods, DelayedArray (>= 0.5.32), rhdf5

Imports utils, tools, BiocGenerics (>= 0.25.1), S4Vectors, IRanges

Suggests h5vcData, SummarizedExperiment (>= 1.9.6), GenomicRanges, BiocStyle

License Artistic-2.0

Collate utils.R HDF5Array-class.R dump-management.R writeHDF5Array.R saveHDF5SummarizedExperiment.R zzz.R

git_url https://git.bioconductor.org/packages/HDF5Array

git_branch RELEASE_3_7

git_last_commit 3c9aa23

git_last_commit_date 2018-06-20

Date/Publication 2018-10-15

R topics documented:

`HDF5-dump-management` *HDF5 dump management*

Description

A set of utilities to control the location of automatically created HDF5 datasets.

Usage

```
setHDF5DumpDir(dir)
setHDF5DumpFile(filepath)
setHDF5DumpName(name)
setHDF5DumpCompressionLevel(level=6L)

getHDF5DumpDir()
getHDF5DumpFile(for.use=FALSE)
getHDF5DumpName(for.use=FALSE)
getHDF5DumpCompressionLevel()

lsHDF5DumpFile()

showHDF5DumpLog()

## For developers:
getHDF5DumpChunkDim(dim, type, ratio=75)
appendDatasetCreationToHDF5DumpLog(filepath, name, dim, type,
                                   chunkdim, level)
```

Arguments

`dir` The path (as a single string) to the current *HDF5 dump directory*, that is, to the (new or existing) directory where *HDF5 dump files* with automatic names will be created. This is ignored if the user specified an *HDF5 dump file* with `setHDF5DumpFile`. If `dir` is missing, then the *HDF5 dump directory* is set back to its default value i.e. to some directory under `tempdir()` (call `getHDF5DumpDir()` to get the exact path).

`filepath` For `setHDF5DumpFile`: The path (as a single string) to the current *HDF5 dump file*, that is, to the (new or existing) HDF5 file where the *next automatic HDF5 datasets* will be written. If `filepath` is missing, then a new file with an automatic name will be created (in `getHDF5DumpDir()`) and used for each new dataset.

For `appendDatasetCreationToHDF5DumpLog`: See the Note TO DEVELOPERS below.

`name` For `setHDF5DumpName`: The name of the *next automatic HDF5 dataset* to be written to the current *HDF5 dump file*.

For `appendDatasetCreationToHDF5DumpLog`: See the Note TO DEVELOPERS below.

`level` For `setHDF5DumpCompressionLevel`: The compression level to use for writting *automatic HDF5 datasets* to disk. See the `level` argument in `?rhdf5::h5createDataset` (in the **rhdf5** package) for more information about this.

For appendDatasetCreationToHDF5DumpLog: See the Note TO DEVELOPERS below.

for.use Whether the returned file or dataset name is for use by the caller or not. See below for the details.

dim The dimensions of the HDF5 dataset to be written to disk, that is, an integer vector of length one or more giving the maximal indices in each dimension. See the dims argument in ?rhdf5::h5createDataset (in the **rhdf5** package) for more information about this.

type The type (a.k.a. storage mode) of the data to be written to disk. Can be obtained with type() on an array-like object (which is equivalent to storage.mode() or typeof() on an ordinary array). This is typically what an application writing datasets to the *HDF5 dump* should pass to the storage.mode argument of its call to rhdf5::h5createDataset. See the Note TO DEVELOPERS below for more information.

ratio The number of chunks per block. By default, a ratio of 75 is used.

chunkdim The dimensions of the chunks.

Details

Calling getHDF5DumpFile() and getHDF5DumpName() with no argument should be *informative* only i.e. it's a mean for the user to know where the *next automatic HDF5 dataset* will be written. Since a given file/name combination can be used only once, the user should be careful to not use that combination to explicitely create an HDF5 dataset because that would get in the way of the creation of the *next automatic HDF5 dataset*. See the Note TO DEVELOPERS below if you actually need to use this file/name combination.

lsHDF5DumpFile() is a just convenience wrapper for rhdf5::h5ls(getHDF5DumpFile()).

Value

getHDF5DumpDir returns the absolute path to the directory where *HDF5 dump files* with automatic names will be created. Only meaningful if the user did NOT specify an *HDF5 dump file* with setHDF5DumpFile.

getHDF5DumpFile returns the absolute path to the HDF5 file where the *next automatic HDF5 dataset* will be written.

getHDF5DumpName returns the name of the *next automatic HDF5 dataset*.

getHDF5DumpCompressionLevel returns the compression level currently used for writting *automatic HDF5 datasets* to disk.

showHDF5DumpLog returns the dump log in an invisible data frame.

getHDF5DumpChunkDim returns the dimension of the chunks for the specified ratio. By default a ratio of 75 is used i.e. the *automatic HDF5 dataset* are written to disk using 75 chunks per block.

Note

TO DEVELOPERS:

If your application needs to write its own dataset to the *HDF5 dump* then it should:

1. Get a file/name combination by calling getHDF5DumpFile(for.use=TRUE) and getHDF5DumpName(for.use=TRUE

OPTIONAL Call getHDF5DumpChunkDim(dim, type) (possibly with a non-default ratio) to get reasonable chunk dimensions to use for writting the data to disk. Or choose your own chunk dimensions.

2. Add an entry to the dump log by calling appendDatasetCreationToHDF5DumpLog. Typically, this should be done right after creating the dataset (e.g. with rhdf5::h5createDataset) and before starting to write the data to disk. The values passed to appendDatasetCreationToHDF5DumpLog via the filepath, name, dim, type, chunkdim, and level arguments should be those that were passed to rhdf5::h5createDataset via the file, dataset, dims, storage.mode, chunk, and level arguments, respectively. Note that appendDatasetCreationToHDF5DumpLog uses a lock mechanism so is safe to use in the context of parallel execution.

This is actually what the coercion method to HDF5Array does internally.

See Also

- writeHDF5Array for writting an array-like object to an HDF5 file.
- HDF5Array objects.
- The h5ls function in the **rhdf5** package, on which lsHDF5DumpFile is based.
- type in the **DelayedArray** package.

Examples

```
getHDF5DumpDir()
getHDF5DumpFile()

## Use setHDF5DumpFile() to change the current HDF5 dump file.
## If the specified file exists, then it must be in HDF5 format or
## an error will be raised. If it doesn't exist, then it will be
## created.
#setHDF5DumpFile("path/to/some/HDF5/file")

lsHDF5DumpFile()

a <- array(1:600, c(150, 4))
A <- as(a, "HDF5Array")
lsHDF5DumpFile()
A

b <- array(runif(6000), c(4, 2, 150))
B <- as(b, "HDF5Array")
lsHDF5DumpFile()
B

C <- (log(2 * A + 0.88) - 5)^3 * t(B[ , 1, ])
as(C, "HDF5Array")  # realize C on disk
lsHDF5DumpFile()

## Matrix multiplication is not delayed: the output matrix is realized
## block by block. The current "realization backend" controls where
## realization happens e.g. in memory if set to NULL or in an HDF5 file
## if set to "HDF5Array". See '?realize' in the DelayedArray package for
## more information about "realization backends".
setRealizationBackend("HDF5Array")
m <- matrix(runif(20), nrow=4)
P <- C %*% m
lsHDF5DumpFile()

## See all the HDF5 datasets created in the current session so far:
```

```
showHDF5DumpLog()

## Wrap the call in suppressMessages() if you are only interested in the
## data frame version of the dump log:
dump_log <- suppressMessages(showHDF5DumpLog())
dump_log
```

`HDF5Array-class` *HDF5 datasets as array-like objects*

Description

We provide 2 classes for representing an (on-disk) HDF5 dataset as an array-like object in R:

- HDF5Array: A high-level class HDF5Array that extends DelayedArray. All the operations available on DelayedArray objects work on HDF5Array objects.
- HDF5ArraySeed: A low-level class for pointing to an HDF5 dataset. No operation can be performed directly on an HDF5ArraySeed object. It needs to be wrapped in a DelayedArray or HDF5Array object first. An HDF5Array object is just an HDF5ArraySeed object wrapped in a DelayedArray object.

Usage

```
## Constructor functions:
HDF5Array(filepath, name, type=NA)
HDF5ArraySeed(filepath, name, type=NA)
```

Arguments

`filepath` The path (as a single character string) to the HDF5 file where the dataset is located.

`name` The name of the dataset in the HDF5 file.

`type` `NA` or the *R atomic type* (specified as a single string) corresponding to the type of the HDF5 dataset.

Value

An HDF5Array object for `HDF5Array()`.

An HDF5ArraySeed object for `HDF5ArraySeed()`.

See Also

- DelayedArray objects.
- DelayedArray-utils for common operations on DelayedArray objects.
- `writeHDF5Array` for writting an array-like object to an HDF5 file.
- HDF5-dump-management for controlling the location of automatically created HDF5 datasets.
- `saveHDF5SummarizedExperiment` and `loadHDF5SummarizedExperiment` in this package (the **HDF5Array** package) for saving/loading an HDF5-based SummarizedExperiment object to/from disk.
- `h5ls` in the **rhdf5** package.
- The **rhdf5** package on top of which HDF5Array objects are implemented.
- array objects in base R.

Examples

```
## ---------------------------------------------------------------------
## CONSTRUCTION
## ---------------------------------------------------------------------
library(rhdf5)
library(h5vcData)

tally_file <- system.file("extdata", "example.tally.hfs5",
                          package="h5vcData")
h5ls(tally_file)

## Pick up "Coverages" dataset for Human chromosome 16:
cov0 <- HDF5Array(tally_file, "/ExampleStudy/16/Coverages")
cov0

## ---------------------------------------------------------------------
## dim/dimnames
## ---------------------------------------------------------------------
dim(cov0)

dimnames(cov0)
dimnames(cov0) <- list(paste0("s", 1:6), c("+", "-"))
dimnames(cov0)

## ---------------------------------------------------------------------
## SLICING (A.K.A. SUBSETTING)
## ---------------------------------------------------------------------
cov1 <- cov0[ , , 29000001:29000007]
cov1

dim(cov1)
as.array(cov1)
stopifnot(identical(dim(as.array(cov1)), dim(cov1)))
stopifnot(identical(dimnames(as.array(cov1)), dimnames(cov1)))

cov2 <- cov0[ , "+", 29000001:29000007]
cov2
as.matrix(cov2)

## ---------------------------------------------------------------------
## SummarizedExperiment OBJECTS WITH DELAYED ASSAYS
## ---------------------------------------------------------------------

## DelayedArray objects can be used inside a SummarizedExperiment object
## to hold the assay data and to delay operations on them.

library(SummarizedExperiment)

pcov <- cov0[ , 1, ]  # coverage on plus strand
mcov <- cov0[ , 2, ]  # coverage on minus strand

nrow(pcov)  # nb of samples
ncol(pcov)  # length of Human chromosome 16

## The convention for a SummarizedExperiment object is to have 1 column
## per sample so first we need to transpose 'pcov' and 'mcov':
```

```
pcov <- t(pcov)
mcov <- t(mcov)
se <- SummarizedExperiment(list(pcov=pcov, mcov=mcov))
se
stopifnot(validObject(se, complete=TRUE))

## A GPos object can be used to represent the genomic positions along
## the dataset:
gpos <- GPos(GRanges("16", IRanges(1, nrow(se))))
gpos
rowRanges(se) <- gpos
se
stopifnot(validObject(se))
assays(se)$pcov
assays(se)$mcov
```

saveHDF5SummarizedExperiment

Save/load an HDF5-based SummarizedExperiment object

Description

saveHDF5SummarizedExperiment and loadHDF5SummarizedExperiment can be used to save/load an HDF5-based SummarizedExperiment object to/from disk.

Usage

```
saveHDF5SummarizedExperiment(x, dir="my_h5_se", replace=FALSE,
                                chunkdim=NULL, level=NULL, verbose=FALSE)
loadHDF5SummarizedExperiment(dir="my_h5_se")
```

Arguments

x
: A SummarizedExperiment object.

dir
: The path (as a single string) to the directory where to save the HDF5-based SummarizedExperiment object or to load it from. When saving, the directory will be created so should not already exist, unless replace is set to TRUE.

replace
: If directory dir already exists, should it be replaced with a new one? The content of the existing directory will be lost!

chunkdim, level
: The dimensions of the chunks and the compression level to use for writting the assay data to disk. Passed to the internal calls to writeHDF5Array. See ?writeHDF5Array for more information.

verbose
: Set to TRUE to make the function display progress.

Details

These functions use functionalities from the **SummarizedExperiment** package internally and so require this package to be installed.

saveHDF5SummarizedExperiment creates the directory specified thru the dir argument and then populates it with the HDF5 datasets (one per assay in x) plus a serialized version of x that contains pointers to these datasets. This directory provides a self-contained HDF5-based representation of x that can then be loaded back in R with loadHDF5SummarizedExperiment. Note that this directory is *relocatable* i.e. it can be moved (or copied) to a different place, on the same or a different computer, before calling loadHDF5SummarizedExperiment on it. For convenient sharing with collaborators, it is suggested to turn it into a tarball (with Unix command tar), or zip file, before the transfer. Please keep in mind that saveHDF5SummarizedExperiment and loadHDF5SummarizedExperiment don't know how to produce/read tarballs or zip files at the moment, so the process of packaging/extracting the tarball or zip file is entirely the user responsibility. It is typically done from outside R.

Finally please note that, depending on the size of the data to write to disk and the performance of the disk, saveHDF5SummarizedExperiment can take a long time to complete. Use verbose=TRUE to see its progress.

loadHDF5SummarizedExperiment is generally very fast, even if the assay data is big, because all the assays in the returned object are HDF5Array objects pointing to the on-disk HDF5 datasets located in dir. HDF5Array objects are typically light-weight in memory.

Value

saveHDF5SummarizedExperiment returns an invisible SummarizedExperiment object where all the assays are HDF5Array objects pointing to the HDF5 datasets saved in dir. It's in fact the same obect as the object that would be returned by calling loadHDF5SummarizedExperiment on dir.

Author(s)

Hervé Pagès

See Also

- SummarizedExperiment and RangedSummarizedExperiment objects in the **SummarizedExperiment** package.
- The writeHDF5Array function which saveHDF5SummarizedExperiment uses internally to write the assay data to disk.

Examples

```
library(SummarizedExperiment)
nrows <- 200; ncols <- 6
counts <- matrix(runif(nrows * ncols, 1, 1e4), nrows)
colData <- DataFrame(Treatment=rep(c("ChIP", "Input"), 3),
                     row.names=LETTERS[1:6])
se0 <- SummarizedExperiment(assays=SimpleList(counts=counts),
                            colData=colData)
se0

## Save 'se0' as an HDF5-based SummarizedExperiment object:
dir <- sub("file", "h5_se0_", tempfile())
h5_se0 <- saveHDF5SummarizedExperiment(se0, dir)
h5_se0
assay(h5_se0, withDimnames=FALSE)  # HDF5Matrix object

h5_se0b <- loadHDF5SummarizedExperiment(dir)
h5_se0b
```

```
assay(h5_se0b, withDimnames=FALSE)  # HDF5Matrix object

## Sanity checks:
stopifnot(is(assay(h5_se0, withDimnames=FALSE), "HDF5Matrix"))
stopifnot(all(DelayedArray(assay(se0)) == assay(h5_se0)))
stopifnot(is(assay(h5_se0b, withDimnames=FALSE), "HDF5Matrix"))
stopifnot(all(DelayedArray(assay(se0)) == assay(h5_se0b)))

## ---------------------------------------------------------------------
## More sanity checks
## ---------------------------------------------------------------------

## Make a copy of directory 'dir':
somedir <- sub("file", "somedir", tempfile())
dir.create(somedir)
file.copy(dir, somedir, recursive=TRUE)
dir2 <- list.files(somedir, full.names=TRUE)

## 'dir2' contains a copy of 'dir'. Call loadHDF5SummarizedExperiment()
## on it.
h5_se0c <- loadHDF5SummarizedExperiment(dir2)

stopifnot(is(assay(h5_se0c, withDimnames=FALSE), "HDF5Matrix"))
stopifnot(all(DelayedArray(assay(se0)) == assay(h5_se0c)))
```

writeHDF5Array *Write an array-like object to an HDF5 file*

Description

A function for writing an array-like object to an HDF5 file.

Usage

```
writeHDF5Array(x, filepath=NULL, name=NULL, chunkdim=NULL, level=NULL,
                  verbose=FALSE)
```

Arguments

x	The array-like object to write to an HDF5 file.

If x is a DelayedArray object, writeHDF5Array *realizes* it on disk, that is, all the delayed operations carried by the object are executed while the object is written to disk. See "On-disk realization of a DelayedArray object as an HDF5 dataset" section below for more information.

filepath	NULL or the path (as a single string) to the (new or existing) HDF5 file where to write the dataset. If NULL, then the dataset will be written to the current *HDF5 dump file* i.e. the path returned by getHDF5DumpFile will be used.

name	NULL or the name of the HDF5 dataset to write. If NULL, then the name returned by getHDF5DumpName will be used.

chunkdim	The dimensions of the chunks to use for writing the data to disk. By default, getHDF5DumpChunkDim(dim(x), type(x)) is used. See ?getHDF5DumpChunkDim for more information.

`level` The compression level to use for writing the data to disk. By default, `getHDF5DumpCompressionLevel` is used. See ?`getHDF5DumpCompressionLevel` for more information.

`verbose` Set to `TRUE` to make the function display progress.

Details

Please note that, depending on the size of the data to write to disk and the performance of the disk, `writeHDF5Array` can take a long time to complete. Use `verbose=TRUE` to see its progress.

Use `setHDF5DumpFile` and `setHDF5DumpName` to control the location of automatically created HDF5 datasets.

Value

An HDF5Array object pointing to the newly written HDF5 dataset on disk.

On-disk realization of a DelayedArray object as an HDF5 dataset

When passed a DelayedArray object, `writeHDF5Array` *realizes* it on disk, that is, all the delayed operations carried by the object are executed on-the-fly while the object is written to disk. This uses a block-processing strategy so that the full object is not realized at once in memory. Instead the object is processed block by block i.e. the blocks are realized in memory and written to disk one at a time.

In other words, `writeHDF5Array(x, ...)` is semantically equivalent to `writeHDF5Array(as.array(x), ...)`, except that `as.array(x)` is not called because this would realize the full object at once in memory.

See ?`DelayedArray` for general information about DelayedArray objects.

See Also

- HDF5Array objects.
- `saveHDF5SummarizedExperiment` and `loadHDF5SummarizedExperiment` in this package (the **HDF5Array** package) for saving/loading an HDF5-based SummarizedExperiment object to/from disk.
- HDF5-dump-management for controlling the location of automatically created HDF5 datasets.
- `h5ls` in the **rhdf5** package.

Examples

```
library(rhdf5)
library(h5vcData)

tally_file <- system.file("extdata", "example.tally.hfs5",
                          package="h5vcData")
h5ls(tally_file)

cov0 <- HDF5Array(tally_file, "/ExampleStudy/16/Coverages")

cov1 <- cov0[ , , 29000001:29000007]

out_file <- tempfile()
writeHDF5Array(cov1, out_file, "cov1")
h5ls(out_file)
```

Index