Package 'hdd'

November 6, 2019

Type Package

Title Easy Manipulation of Out of Memory Data Sets

Version 0.1.0

Imports fst, utils, readr

Depends data.table

Suggests knitr, rmarkdown

VignetteBuilder knitr

Description Hard drive data: Class of data allowing the easy importation/manipulation of out of memory data sets. The data sets are located on disk but look like in-memory, the syntax for manipulation is similar to 'data.table'. Operations are performed ``chunk-wise'' behind the scene.

License GPL-3

Encoding UTF-8

LazyData true

RoxygenNote 6.1.1

NeedsCompilation no

Author Laurent Berge [aut, cre]

Maintainer Laurent Berge <laurent.berge@uni.lu>

Repository CRAN

Date/Publication 2019-11-06 15:00:03 UTC

R topics documented:

`hdd-package` *Easy manipulation of out of memory data sets*

Description

hdd offers a class of data, hard drive data, allowing the easy importation/manipulation of out of memory data sets. The data sets are located on disk but look like in-memory, the syntax for manipulation is similar to `data.table`. Operations are performed "chunk-wise" behind the scene.

Details

The functions for importations is `txt2hdd`. The loading of a hdd data set is done with `hdd` and the data is extracted with `sub-.hdd` which has a `data.table` syntax. You can alternatively create a hdd data set with `hdd_slice`. Other utilities include `hdd_merge`, or `peek` to have a quick look into a text file containing data.

Author(s)

Laurent Berge

`dim.hdd` *Dimension of a HDD object*

Description

Gets the dimension of a hard drive data set (HDD).

Usage

```
## S3 method for class 'hdd'
dim(x)
```

Arguments

x	A HDD object.

Value

It returns a vector of length 2 containing the number of rows and the number of columns of the HDD object.

Author(s)

Laurent Berge

Examples

```
# Toy example with iris data
iris_path = tempfile()
fwrite(iris, iris_path)

# destination path
hdd_path = tempfile()

# reading the text file with 50 rows chunks:
txt2hdd(iris_path, dirDest = hdd_path, rowsPerChunk = 50)

# creating a HDD object
base_hdd = hdd(hdd_path)

# Summary information on the whole data set
summary(base_hdd)

# Looking at it like a regular data.frame
print(base_hdd)
dim(base_hdd)
names(base_hdd)
```

---

guess_col_types *Guesses the columns types of a file*

Description

This function is a facility to guess the column types of a text document. It returns columns formatted a la readr.

Usage

```
guess_col_types(dt_or_path, col_names, n = 10000)
```

Arguments

`dt_or_path`	Either a data frame or a path.
`col_names`	Optional: the vector of names of the columns, if not contained in the file. Must match the number of columns in the file.
`n`	Number of observations used to make the guess. By default, `n = 100000`.

Details

The guessing of the column types is based on the 10,000 (set with argument `n`) first rows.

Note that by default, columns that are found to be integers are imported as double (in want of integer64 type in readr). Note that for large data sets, sometimes integer-like identifiers can be larger than 16 digits: in these case you must import them as character not to lose information.

Value

It returns a `cols` object a la `readr`.

Author(s)

Laurent Berge

See Also

See `peek` to have a convenient look at the first lines of a text file. See `guess_delim` to guess the delimiter of a text data set. See `guess_col_types` to guess the column types of a text data set.

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

Examples

```
# Example with the iris data set
iris_path = tempfile()
fwrite(iris, iris_path)

# returns a readr columns set:
guess_col_types(iris_path)
```

`guess_delim` *Guesses the delimiter of a text file*

Description

This function uses `fread` to guess the delimiter of a text file.

Usage

```
guess_delim(path)
```

Arguments

`path`	The path to a text file containing a rectangular data set.

Value

It returns a character string of length 1: the delimiter.

Author(s)

Laurent Berge

See Also

See `peek` to have a convenient look at the first lines of a text file. See `guess_delim` to guess the delimiter of a text data set. See `guess_col_types` to guess the column types of a text data set.

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

Examples

```
# Example with the iris data set
iris_path = tempfile()
fwrite(iris, iris_path)

guess_delim(iris_path)
```

`hdd` *Hard drive data set*

Description

This function connects to a hard drive data set (HDD). You can access the hard drive data in a similar way to a `data.table`.

Usage

```
hdd(dir)
```

Arguments

`dir`	The directory where the hard drive data set is.

Details

HDD has been created to deal with out of memory data sets. The data set exists in the hard drive, split in multiple files – each file being workable in memory.

You can perform extraction and manipulation operations as with a regular data set with `sub-.hdd`. Each operation is performed chunk-by-chunk behind the scene.

In terms of performance, working with complete data sets in memory will always be faster. This is because read/write operations on disk are order of magnitude slower than read/write in memory. However, this might be the only way to deal with out of memory data.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data
iris_path = tempfile()
fwrite(iris, iris_path)
```

```
# destination path
hdd_path = tempfile()

# reading the text file with 50 rows chunks:
txt2hdd(iris_path, dirDest = hdd_path, rowsPerChunk = 50)

# creating a HDD object
base_hdd = hdd(hdd_path)

# Summary information on the whole data set
summary(base_hdd)

# Looking at it like a regular data.frame
print(base_hdd)
dim(base_hdd)
names(base_hdd)
```

`hdd_merge` *Merges data to a HDD file*

Description

This function merges in-memory/HDD data to a HDD file.

Usage

```
hdd_merge(x, y, newfile, chunkMB, rowsPerChunk, all = FALSE,
  all.x = all, all.y = all, allow.cartesian = FALSE,
  replace = FALSE, verbose)
```

Arguments

`x`	A HDD object or a `data.frame`.
`y`	A data set either a data.frame of a HDD object.
`newfile`	Destination of the result, i.e., a destination folder that will receive the HDD data.
`chunkMB`	Numeric, default is missing. If provided, the data 'x' is split in chunks of 'chunkMB' MB and the merge is applied chunkwise.
`rowsPerChunk`	Integer, default is missing. If provided, the data 'x' is split in chunks of 'rowsPerChunk' rows and the merge is applied chunkwise.
`all`	Default is `FALSE`.
`all.x`	Default is `all`.
`all.y`	Default is `all`.
`allow.cartesian`	Logical: whether to allow cartesian merge. Defaults to `FALSE`.

`replace`	Default is `FALSE`: if the destination folder already contains data, whether to replace it.
`verbose`	Numeric. Whether information on the advancement should be displayed. If equal to 0, nothing is displayed. By default it is equal to 1 if the size of `x` is greater than 1GB.

Details

If `x` (resp `y`) is a HDD object, then the merging will be operated chunkwise, with the original chunks of the objects. To change the size of the chunks for `x`: you can use the argument `chunkMB` or `rowsPerChunk`.

To change the chunk size of `y`, you can rewrite `y` with a new chunk size using `write_hdd`.

Note that the merging operation could also be achieved with `hdd_slice` (although it would require setting up a ad hoc function).

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data

# Cartesian merge example
iris_bis = iris
names(iris_bis) = c(paste0("x_", 1:4), "species_bis")
# We must have a common key on which to merge
iris_bis$id = iris$id = 1

# merge, we chunk 'x' by 50 rows
hdd_path = tempfile()
hdd_merge(iris, iris_bis, newfile = hdd_path,
  rowsPerChunk = 50, allow.cartesian = TRUE)

base_merged = hdd(hdd_path)
summary(base_merged)
print(base_merged)
```

hdd_setkey *Sorts HDD objects*

Description

This function sets a key to a HDD file. It creates a copy of the HDD file sorted by the key. Note that the sorting process is very time consuming.

Usage

```
hdd_setkey(x, key, newfile, chunkMB = 500, replace = FALSE,
  verbose = 1)
```

Arguments

`x`	A hdd file.
`key`	A character vector of the keys.
`newfile`	Destination of the result, i.e., a destination folder that will receive the HDD data.
`chunkMB`	The size of chunks used to sort the data. Default is 500MB. The bigger this number the faster the sorting is (depends on your memory available though).
`replace`	Default is `FALSE`: if the destination folder already contains data, whether to replace it.
`verbose`	Numeric, default is 1. Whether to display information on the advancement of the algorithm. If equal to 0, nothing is displayed.

Details

This function is provided for convenience reason: it does the job of sorting the data and ensuring consistency across files, but it is very slow since it involves copying several times the entire data set. To be used parsimoniously.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data

# Creating HDD data to be sorted
hdd_path = tempfile() # => folder where the data will be saved
write_hdd(iris, hdd_path)
# Let's add data to it
for(i in 1:10) write_hdd(iris, hdd_path, add = TRUE)

base_hdd = hdd(hdd_path)
summary(base_hdd)

# Sorting by Sepal.Width
hdd_sorted = tempfile()
# we use a very small chunkMB to show how the function works
hdd_setkey(base_hdd, key = "Sepal.Width",
   newfile = hdd_sorted, chunkMB = 0.010)


base_hdd_sorted = hdd(hdd_sorted)
summary(base_hdd_sorted) # => additional line "Sorted by:"
print(base_hdd_sorted)

# Sort with two keys:
hdd_sorted = tempfile()
# we use a very small chunkMB to show how the function works
hdd_setkey(base_hdd, key = c("Species", "Sepal.Width"),
   newfile = hdd_sorted, chunkMB = 0.010)


base_hdd_sorted = hdd(hdd_sorted)
summary(base_hdd_sorted)
print(base_hdd_sorted)
```

`hdd_slice` *Applies a function to slices of data to create a HDD data set*

Description

This function is useful to apply complex R functions to large data sets (out of memory). It slices the input data, applies the function, then saves each chunk into a hard drive folder. This can then be a HDD data set.

Usage

```
hdd_slice(x, fun, dir, chunkMB = 500, rowsPerChunk, replace = FALSE,
  verbose = 1, ...)
```

Arguments

`x`	A data set (data.frame, HDD).
`fun`	A function to be applied to slices of the data set. The function must return a data frame like object.
`dir`	The destination directory where the data is saved.
`chunkMB`	The size of the slices, default is 500MB. That is: the function `fun` is applied to each 500Mb of data `x`. If the function creates a lot of additional information, you may want this number to go down. On the other hand, if the function reduces the information you may want this number to go up. In the end it will depend on the amount of memory available.
`rowsPerChunk`	Integer, default is missing. Alternative to the argument `chunkMB`. If provided, the functions will be applied to chunks of `rowsPerChunk` of `x`.
`replace`	Whether all information on the destination directory should be erased beforehand. Default is `FALSE`.
`verbose`	Integer, defaults to 1. If greater than 0 then the progress is displayed.
`...`	Other parameters to be passed to `fun`.

Details

This function splits the original data into several slices and then apply a function to each of them, saving the results into a HDD data set.

You can perform merging operations with `hdd_slice`, but for regular merges not that you have the function `hdd_merge` that may prove more convenient (not need to write a ad hoc function).

Value

It doesn't return anything, the output is a "hard drive data" saved in the hard drive.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data.
# Say you want to perform a cartesian merge
# If the results of the function is out of memory
# you can use hdd_slice (not the case for this example)

# preparing the cartesian merge
iris_bis = iris
names(iris_bis) = c(paste0("x_", 1:4), "species_bis")


fun_cartesian = function(x){
# Note that x is treated as a data.table
# => we need the argument allow.cartesian
merge(x, iris_bis, allow.cartesian = TRUE)
}

hdd_result = tempfile() # => folder where results are saved
hdd_slice(iris, fun_cartesian, dir = hdd_result, rowsPerChunk = 30)

# Let's look at the result
base_hdd = hdd(hdd_result)
summary(base_hdd)
head(base_hdd)
```

`names.hdd` *Variables names of a HDD object*

Description

Gets the variable names of a hard drive data set (HDD).

Usage

```
## S3 method for class 'hdd'
names(x)
```

Arguments

x	A `HDD` object.

Value

A character vector.

Author(s)

Laurent Berge

See Also

See hdd, sub-.hdd and cash-.hdd for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see txt2hdd.

See hdd_slice to apply functions to chunks of data (and create HDD objects) and hdd_merge to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see write_hdd.

To display general information from HDD objects: origin, summary.hdd, print.hdd, dim.hdd and names.hdd.

Examples

```
# Toy example with iris data
iris_path = tempfile()
fwrite(iris, iris_path)

# destination path
hdd_path = tempfile()

# reading the text file with 50 rows chunks:
txt2hdd(iris_path, dirDest = hdd_path, rowsPerChunk = 50)

# creating a HDD object
base_hdd = hdd(hdd_path)

# Summary information on the whole data set
summary(base_hdd)

# Looking at it like a regular data.frame
print(base_hdd)
dim(base_hdd)
names(base_hdd)
```

origin *Extracts the origin of a HDD object*

Description

Use this function to extract the information on how the HDD data set was created.

Usage

```
origin(x)
```

Arguments

x	A HDD object.

Details

Each HDD lives on disk and a "_hdd.txt" is always present in the folder containing summary information. The function `origin` extracts the log from this information file.

Value

A character vector, if the HDD data set has been created with several instances of `write_hdd` its length will be greater than 1.

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data

hdd_path = tempfile()
write_hdd(iris, hdd_path, rowsPerChunk = 20)

base_hdd = hdd(hdd_path)
origin(base_hdd)

# Let's add something
write_hdd(head(iris), hdd_path, add = TRUE)
write_hdd(iris, hdd_path, add = TRUE, rowsPerChunk = 50)

base_hdd = hdd(hdd_path)
origin(base_hdd)
```

peek *Peek into a text file*

Description

This function looks at the first elements of a file, format it into a data frame and displays it. It can also just show the first lines of the file without formatting into a DF.

Usage

```
peek(path, onlyLines = FALSE, n, view = TRUE)
```

Arguments

`path`	Path linking to the text file.
`onlyLines`	Default is `FALSE`. If `TRUE`, then the first `n` lines are directly displayed without formatting.
`n`	Integer. The number of lines to extract from the file. Default is 100 or 5 if `onlyLine = TRUE`.
`view`	Logical, default it `TRUE`: whether the data should be displayed on the viewer. Only when `onlyLines = FALSE`.

Value

Returns the data invisibly.

Author(s)

Laurent Berge

See Also

See `peek` to have a convenient look at the first lines of a text file. See `guess_delim` to guess the delimiter of a text data set. See `guess_col_types` to guess the column types of a text data set.

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

Examples

```
# Example with the iris data set
iris_path = tempfile()
fwrite(iris, iris_path)


# The first lines of the text file on viewer
peek(iris_path)
```

```
# displaying the first lines:
peek(iris_path, onlyLines = TRUE)

# only getting the data from the first observations
base = peek(iris_path, view = FALSE)
head(base)
```

---

print.hdd *Print method for HDD objects*

Description

This functions displays the first and last lines of a hard drive data set (HDD).

Usage

```
## S3 method for class 'hdd'
print(x, ...)
```

Arguments

`x`	A `HDD` object.
`...`	Not currently used.

Details

Returns the first and last 3 lines of a HDD object. Also formats the values displayed on screen (typically: add commas to increase the readability of large integers).

Value

Nothing is returned.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data
iris_path = tempfile()
fwrite(iris, iris_path)

# destination path
hdd_path = tempfile()

# reading the text file with 50 rows chunks:
txt2hdd(iris_path, dirDest = hdd_path, rowsPerChunk = 50)

# creating a HDD object
base_hdd = hdd(hdd_path)

# Summary information on the whole data set
summary(base_hdd)

# Looking at it like a regular data.frame
print(base_hdd)
dim(base_hdd)
names(base_hdd)
```

`readfst` *Read fst or HDD files as DT*

Description

This is the function `read_fst` but with automatic conversion to data.table. It also allows to read hdd data.

Usage

```
readfst(path, columns = NULL, from = 1, to = NULL, confirm = FALSE)
```

Arguments

`path`	Path to `fst` file – or path to hdd data. For hdd files, there is a
`columns`	Column names to read. The default is to read all columns. Ignored for hdd files.
`from`	Read data starting from this row number. Ignored for hdd files.
`to`	Read data up until this row number. The default is to read to the last row of the stored data set. Ignored for hdd files.
`confirm`	If the hdd file is larger than the variable `getHdd_extract.cap()`, then by default an error is raised. To anyway read the data, use `confirm = TRUE`. You can set the data cap with the function `setHdd_extract.cap`, the default being 1GB.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with the iris data set

# writing a hdd file
hdd_path = tempfile()
write_hdd(iris, hdd_path, rowsPerChunk = 30)

# reading the full data in memory
base_mem = readfst(hdd_path)

# is equivalent to:
base_hdd = hdd(hdd_path)
base_mem_bis = base_hdd[]
```

`setHdd_extract.cap` *Sets/gets the size cap when extracting hdd data*

Description

Sets/gets the default size cap when extracting HDD variables with `cash-.hdd` or when importing full HDD data sets with `readfst`.. If the size exceeds the cap, then an error is raised, which can be bypassed by using the argument `confirm`.

Usage

```
setHdd_extract.cap(sizeMB = 1000)

getHdd_extract.cap()
```

Arguments

`sizeMB`	Size cap in MB. Default to 1000.

Value

The size cap, a numeric scalar.

Examples

```
# Toy example with iris data
# We first create a hdd dataset with approx. 100KB
hdd_path = tempfile() # => folder where the data will be saved
write_hdd(iris, hdd_path)
for(i in 1:10) write_hdd(iris, hdd_path, add = TRUE)

base_hdd = hdd(hdd_path)
summary(base_hdd) # => 11 files

# we can extract the data from the 11 files with '$':
pl = base_hdd$Sepal.Length


#
# Illustration of the protection mechanism:
#

# By default you cannot extract a variable with '$'
# when its size would be too large (default is greater than 1000MB)
# You can set the cap with setHdd_extract.cap.

# Following code raises an error:
setHdd_extract.cap(sizeMB = 0.005) # new cap of 5KB
pl = base_hdd$Sepal.Length

# To extract the variable without changing the cap:
pl = base_hdd[, Sepal.Length] # => no size control is performed

# Resetting the default cap
setHdd_extract.cap()
```

`summary.hdd` *Summary information for HDD objects*

Description

Provides summary information – i.e. dimension, size on disk, path, number of slices – of hard drive data sets (HDD).

Usage

```
## S3 method for class 'hdd'
summary(object, ...)
```

Arguments

`object`	A HDD object.
`...`	Not currently used.

Details

Displays concisely general information on the HDD object: its size on disk, the number of files it is made of, its location on disk and the number of rows and columns.

Note that each HDD object contain the text file "_hdd.txt" in their folder also containing this information.

To obtain how the HDD object was constructed, use function `origin`.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data
iris_path = tempfile()
fwrite(iris, iris_path)

# destination path
hdd_path = tempfile()

# reading the text file with 50 rows chunks:
txt2hdd(iris_path, dirDest = hdd_path, rowsPerChunk = 50)

# creating a HDD object
base_hdd = hdd(hdd_path)

# Summary information on the whole data set
```

```
summary(base_hdd)

# Looking at it like a regular data.frame
print(base_hdd)
dim(base_hdd)
names(base_hdd)
```

`txt2hdd` *Transforms text data into a HDD file*

Description

Imports text data and saves it into a HDD file. It uses `read_delim_chunked` to extract the data. It also allows to preprocess the data.

Usage

```
txt2hdd(path, dirDest, chunkMB = 500, rowsPerChunk, col_names, col_types,
  nb_skip, delim, preprocessfun, replace = FALSE, verbose, ...)
```

Arguments

`path`	Path where the data is.
`dirDest`	The destination directory, where the new HDD data should be saved.
`chunkMB`	The chunk sizes in MB, defaults to 500MB. Instead of using this argument, you can alternatively use the argument `rowsPerChunk` which decides the size of chunks in terms of lines.
`rowsPerChunk`	Number of rows per chunk. By default it is missing: its value is deduced from argument `chunkMB` and the size of the file. If provided, replaces any value provided in `chunkMB`.
`col_names`	The column names, by default is uses the ones of the data set. If the data set lacks column names, you must provide them.
`col_types`	The column types, in the `readr` fashion. You can use `guess_col_types` to find them.
`nb_skip`	Number of lines to skip.
`delim`	The delimiter. By default the function tries to find the delimiter, but sometimes it fails.
`preprocessfun`	A function that is applied to the data before saving. Default is missing. Note that if a function is provided, it MUST return a data.frame, anything other than data.frame is ignored.
`replace`	If the destination directory already exists, you need to set the argument `replace=TRUE` to overwrite all the HDD files in it.

`verbose`	Integer. If verbose > 0, then the evolution of the importing process is reported. By default: equal to 1 when the expected number of chunks is greater than 1.
`...`	Other arguments to be passed to `read_delim_chunked`, `quote = ""` can be interesting sometimes.

Details

This function uses `read_delim_chunked` from `readr` to read a large text file per chunk, and generate a HDD data set.

Since the main function for importation uses `readr`, the column specification must also be in readr's style (namely `cols` or `cols_only`).

By default a guess of the column types is made on the first 10,000 rows. The guess is the application of `guess_col_types` on these rows.

Note that by default, columns that are found to be integers are imported as double (in want of integer64 type in readr). Note that for large data sets, sometimes integer-like identifiers can be larger than 16 digits: in these case you must import them as character not to lose information.

The delimiter is found with the function `guess_delim`, which uses the guessing from `fread`. Note that fixed width delimited files are not supported.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data

# we create a text file on disk
iris_path = tempfile()
fwrite(iris, iris_path)

# destination path
hdd_path = tempfile()
# reading the text file with HDD, with approx. 50 rows per chunk:
txt2hdd(iris_path, dirDest = hdd_path, rowsPerChunk = 50)

base_hdd = hdd(hdd_path)
```

```
summary(base_hdd)

# Same example with preprocessing
sl_keep = sort(unique(sample(iris$Sepal.Length, 40)))
fun = function(x){
# we keep only some observations & vars + renaming
res = x[Sepal.Length %in% sl_keep, .(sl = Sepal.Length, Species)]
# we create some variables
res[, sl2 := sl**2]
res
}
# reading with preprocessing
hdd_path_preprocess = tempfile()
txt2hdd(iris_path, hdd_path_preprocess,
preprocessfun = fun, rowsPerChunk = 50)

base_hdd_preprocess = hdd(hdd_path_preprocess)
summary(base_hdd_preprocess)
```

`write_hdd` *Saves or appends a data set into a HDD file*

Description

This function saves in-memory/HDD data sets into HDD repositories. Useful to append several data sets.

Usage

```
write_hdd(x, dir, chunkMB = Inf, rowsPerChunk, compress = 50,
  add = FALSE, replace = FALSE, showWarning, ...)
```

Arguments

`x`	A data set.
`dir`	The HDD repository, i.e. the directory where the HDD data is.
`chunkMB`	If the data has to be split in several files of `chunkMB` sizes. Default is `Inf`.
`rowsPerChunk`	Integer, default is missing. Alternative to the argument `chunkMB`. If provided, the data will be split in several files of `rowsPerChunk` rows.
`compress`	Compression rate to be applied by `write_fst`. Default is 50.
`add`	Should the file be added to the existing repository? Default is `FALSE`.
`replace`	If `add = FALSE`, should any existing document be replaced? Default is `FALSE`.
`showWarning`	If the data `x` has no observation, then a warning is raised if `showWarning = TRUE`. By default, it occurs only if `write_hdd` is NOT called within a function.
`...`	Not currently used.

Details

Creating a HDD data set with this function always create an additional file named "_hdd.txt" in the HDD folder. This file contains summary information on the data: the number of rows, the number of variables, the first five lines and a log of how the HDD data set has been created. To access the log directly from R, use the function `origin`.

Author(s)

Laurent Berge

See Also

See `hdd`, `sub-.hdd` and `cash-.hdd` for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see `txt2hdd`.

See `hdd_slice` to apply functions to chunks of data (and create HDD objects) and `hdd_merge` to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see `write_hdd`.

To display general information from HDD objects: `origin`, `summary.hdd`, `print.hdd`, `dim.hdd` and `names.hdd`.

Examples

```
# Toy example with iris data

# Let's create a HDD data set from iris data
hdd_path = tempfile() # => folder where the data will be saved
write_hdd(iris, hdd_path)
# Let's add data to it
for(i in 1:10) write_hdd(iris, hdd_path, add = TRUE)

base_hdd = hdd(hdd_path)
summary(base_hdd) # => 11 files, 1650 lines, 48.7KB on disk

# Let's save the iris data by chunks of 1KB
# we use replace = TRUE to delete the previous data
write_hdd(iris, hdd_path, chunkMB = 0.001, replace = TRUE)

base_hdd = hdd(hdd_path)
summary(base_hdd) # => 8 files, 150 lines, 10.2KB on disk
```

`[.hdd` *Extraction of HDD data*

Description

This function extract data from HDD files, in a similar fashion as data.table but with more arguments.

Usage

```
## S3 method for class 'hdd'
x[index, ..., file, newfile, replace = FALSE,
  all.vars = FALSE]
```

Arguments

`x`	A hdd file.
`index`	An index, you can use `.N` and variable names, like in data.table.
`...`	Other components of the extraction to be passed to `data.table`.
`file`	Which file to extract from? (Remember hdd data is split in several files.) You can use `.N`.
`newfile`	A destination directory. Default is missing. Should be result of the query be saved into a new HDD directory? Otherwise, it is put in memory.
`replace`	Only used if argument `newfile` is not missing: default is `FALSE`. If the `newfile` points to an existing HDD data, then to replace it you must have `replace = TRUE`.
`all.vars`	Logical, default is `FALSE`. By default, if the first argument of `...` is provided (i.e. argument `j`) then only variables appearing in all `...` plus the variable names found in `index` are extracted. If `TRUE` all variables are extracted before any selection is done. (This can be useful when the algorithm getting the variable names gets confused in case of complex queries.)

Details

The extraction of variables look like a regular `data.table` extraction but in fact all operations are made chunk-by-chunk behind the scene.

The extra arguments `file`, `newfile` and `replace` are added to a regular `data.table` call. Argument `file` is used to select the chunks, you can use the special variable `.N` to identify the last chunk.

By default, the operation loads the data in memory. But if the expected size is still too large, you can use the argument `newfile` to create a new HDD data set without size restriction. If a HDD data set already exists in the `newfile` destination, you can use the argument `replace=TRUE` to override it.

Value

Returns a data.table extracted from a HDD file (except if newwfile is not missing).

Author(s)

Laurent Berge

See Also

See hdd, sub-.hdd and cash-.hdd for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see txt2hdd.

See hdd_slice to apply functions to chunks of data (and create HDD objects) and hdd_merge to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see write_hdd.

To display general information from HDD objects: origin, summary.hdd, print.hdd, dim.hdd and names.hdd.

Examples

```
# Toy example with iris data

# First we create a hdd data set to run the example
hdd_path = tempfile()
write_hdd(iris, hdd_path, rowsPerChunk = 40)

# your data set is in the hard drive, in hdd format already.
data_hdd = hdd(hdd_path)

# summary information on the whole file:
summary(data_hdd)

# You can use the argument 'file' to subselect slices.
# Let's have some descriptive statistics of the first slice of HDD
summary(data_hdd[, file = 1])

# It extract the data from the first HDD slice and
# returns a data.table in memory, we then apply summary to it
# You can use the special argument .N, as in data.table.

# the following query shows the first and last lines of
# each slice of the HDD data set:
data_hdd[c(1, .N), file = 1:.N]

# Extraction of observations for which the variable
# Petal.Width is lower than 0.1
data_hdd[Petal.Width < 0.2, ]

# You can apply data.table syntax:
data_hdd[, .(pl = Petal.Length)]

# and create variables
data_hdd[, pl2 := Petal.Length**2]

# You can use the by clause, but then
# the by is applied slice by slice, NOT on the full data set:
data_hdd[, .(mean_pl = mean(Petal.Length)), by = Species]

# If the data you extract does not fit into memory,
# you can create a new HDD file with the argument 'newfile':
hdd_path_new = tempfile()
data_hdd[, pl2 := Petal.Length**2, newfile = hdd_path_new]
# check the result:
data_hdd_bis = hdd(hdd_path_new)
summary(data_hdd_bis)
print(data_hdd_bis)
```

`$.hdd` *Extracts a single variable from a HDD object*

Description

This method extracts a single variable from a hard drive data set (HDD). There is an automatic protection to avoid extracting too large data into memory. The bound is set by the function `setHdd_extract.cap`.

Usage

```
## S3 method for class 'hdd'
x$name
```

Arguments

`x`	A `HDD` object.
`name`	The variable name to be extracted.Note that there is an automatic protection for not trying to import data that would not fit into memory. The extraction cap is set with the function `setHdd_extract.cap`.

Details

By default if the expected size of the variable to extract is greater than the value given by `getHdd_extract.cap` an error is raised. For numeric variables, the expected size is exact. For non-numeric data, the expected size is a guess that considers all the non-numeric variables being of the same size. This may lead to an over or under estimation depending on the cases. In any case, if your variable is large and you don't want to change the extraction cap (`setHdd_extract.cap`), you can still extract the variable with `sub-.hdd` for which there is no such protection.

Note that you cannot create variables with `$`, e.g. like `base_hdd$x_new <-something`. To create variables, use the `[` instead (see `sub-.hdd`).

Value

It returns a vector.

Author(s)

Laurent Berge

See Also

See hdd, sub-.hdd and cash-.hdd for the extraction and manipulation of out of memory data. For importation of HDD data sets from text files: see txt2hdd.

See hdd_slice to apply functions to chunks of data (and create HDD objects) and hdd_merge to merge large files.

To create/reshape HDD objects from memory or from other HDD objects, see write_hdd.

To display general information from HDD objects: origin, summary.hdd, print.hdd, dim.hdd and names.hdd.

Examples

```
# Toy example with iris data
# We first create a hdd dataset with approx. 100KB
hdd_path = tempfile() # => folder where the data will be saved
write_hdd(iris, hdd_path)
for(i in 1:10) write_hdd(iris, hdd_path, add = TRUE)

base_hdd = hdd(hdd_path)
summary(base_hdd) # => 11 files

# we can extract the data from the 11 files with '$':
pl = base_hdd$Sepal.Length


#
# Illustration of the protection mechanism:
#

# By default you cannot extract a variable with '$'
# when its size would be too large (default is greater than 1000MB)
# You can set the cap with setHdd_extract.cap.

# Following code raises an error:
setHdd_extract.cap(sizeMB = 0.005) # new cap of 5KB
pl = base_hdd$Sepal.Length

# To extract the variable without changing the cap:
pl = base_hdd[, Sepal.Length] # => no size control is performed

# Resetting the default cap
setHdd_extract.cap()
```

Index