

# Package ‘disk.frame’

August 24, 2023

**Type** Package

**Title** Larger-than-RAM Disk-Based Data Manipulation Framework

**Version** 0.8.3

**Date** 2023-08-01

**Maintainer** Dai ZJ <zhuojia.dai@gmail.com>

**Description** A disk-based data manipulation tool for working with large-than-RAM datasets. Aims to lower the barrier-to-entry for manipulating large datasets by adhering closely to popular and familiar data manipulation paradigms like 'dplyr' verbs and 'data.table' syntax.

**License** MIT + file LICENSE

**Imports** Rcpp (>= 0.12.13), glue (>= 1.3.1), future.apply (>= 1.3.0), fs (>= 1.3.1), jsonlite (>= 1.6), pryr (>= 0.1.4), stringr (>= 1.4.0), fst (>= 0.8.0), future (>= 1.14.0), data.table (>= 1.12.2), crayon (>= 1.3.4), bigreadr (>= 0.2.0), bit64, benchmarkme, purrr (>= 0.3.2), globals, rlang, arrow

**Depends** R (>= 4.0), dplyr (>= 1.1.0)

**Suggests** nycflights13, magrittr, shiny, LaF, readr, rstudioapi, broom, ggplot2

**LinkingTo** Rcpp

**RoxygenNote** 7.2.3

**Encoding** UTF-8

**URL** <https://diskframe.com>

**BugReports** <https://github.com/DiskFrame/disk.frame/issues>

**NeedsCompilation** yes

**Author** Dai ZJ [aut, cre],  
Jacky Poon [ctb]

**Repository** CRAN

**Date/Publication** 2023-08-24 16:20:10 UTC

**R topics documented:**

add_chunk . . . . .	3
anti_join.disk.frame . . . . .	4
as.data.frame.disk.frame . . . . .	7
as.data.table.disk.frame . . . . .	8
as.disk.frame . . . . .	8
bind_rows.disk.frame . . . . .	9
chunk_summarize . . . . .	10
cmap . . . . .	11
cmap2 . . . . .	13
collect.disk.frame . . . . .	14
colnames . . . . .	15
compute.disk.frame . . . . .	16
create_chunk_mapper . . . . .	16
csv_to_disk.frame . . . . .	17
delete . . . . .	19
df_ram_size . . . . .	19
disk.frame . . . . .	20
disk.frame_to_parquet . . . . .	20
evalparseglue . . . . .	21
find_globals_recursively . . . . .	21
foverlaps.disk.frame . . . . .	22
gen_datatable_synthetic . . . . .	23
get_chunk . . . . .	23
get_chunk_ids . . . . .	24
get_partition_paths . . . . .	25
groups.disk.frame . . . . .	25
head.disk.frame . . . . .	26
is_disk.frame . . . . .	26
merge.disk.frame . . . . .	27
move_to . . . . .	28
nchunks . . . . .	29
nrow . . . . .	29
overwrite_check . . . . .	30
partition_filter . . . . .	31
play . . . . .	31
print.disk.frame . . . . .	32
pull.disk.frame . . . . .	32
purrr_as_mapper . . . . .	33
rbindlist.disk.frame . . . . .	33
rechunk . . . . .	34
recommend_nchunks . . . . .	35
remove_chunk . . . . .	36
sample_frac.disk.frame . . . . .	36
select.disk.frame . . . . .	37
setup_disk.frame . . . . .	38
shard . . . . .	39

`add_chunk` 3

<code>shardkey</code> . . . . .	40
<code>shardkey_equal</code> . . . . .	41
<code>show_ceremony</code> . . . . .	41
<code>split_string_into_df</code> . . . . .	41
<code>srckeep</code> . . . . .	42
<code>summarise.grouped_disk.frame</code> . . . . .	42
<code>tbl_vars.disk.frame</code> . . . . .	43
<code>var_df.chunk_agg.disk.frame</code> . . . . .	44
<code>write_disk.frame</code> . . . . .	45
<code>zip_to_disk.frame</code> . . . . .	46
<code>[].disk.frame</code> . . . . .	47

**Index** 49

---

<code>add_chunk</code>	<i>Add a chunk to the disk.frame</i>
------------------------	--------------------------------------

---

### Description

If no `chunk_id` is specified, then the chunk is added at the end as the largest numbered file, "n.fst".

### Usage

```
add_chunk(df, chunk, chunk_id = NULL, full.names = FALSE, ...)
```

### Arguments

<code>df</code>	the <code>disk.frame</code> to add a chunk to
<code>chunk</code>	a <code>data.frame</code> to be added as a chunk
<code>chunk_id</code>	a numeric number indicating the id of the chunk. If <code>NULL</code> it will be set to the largest <code>chunk_id + 1</code>
<code>full.names</code>	whether the <code>chunk_id</code> name match should be to the full file path not just the file name
<code>...</code>	Passed in the <code>write_fst</code> . E.g. <code>compress</code>

### Details

The function is the preferred way to add a chunk to a `disk.frame`. It performs checks on the types to make sure that the new chunk doesn't have different types to the `disk.frame`.

### Value

`disk.frame`

**Examples**

```

# create a disk.frame
df_path = file.path(tempdir(), "tmp_add_chunk")
diskf = disk.frame(df_path)

# add a chunk to diskf
add_chunk(diskf, cars)
add_chunk(diskf, cars)

nchunks(diskf) # 2

df2 = disk.frame(file.path(tempdir(), "tmp_add_chunk2"))

# add chunks by specifying the chunk_id number; this is especially useful if
# you wish to add multiple chunk in parallel

add_chunk(df2, data.frame(chunk=1), 1)
add_chunk(df2, data.frame(chunk=2), 3)

nchunks(df2) # 2

dir(attr(df2, "path", exact=TRUE))
# [1] "1.fst" "3.fst"

# clean up
delete(diskf)
delete(df2)

```

---

anti\_join.disk.frame *Performs join/merge for disk.frames*

---

**Description**

Performs join/merge for disk.frames

**Usage**

```

## S3 method for class 'disk.frame'
anti_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_anti_join"),
  merge_by_chunk_id = FALSE,
  overwrite = TRUE,
  .progress = FALSE
)

```

```
## S3 method for class 'disk.frame'
full_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_full_join"),
  overwrite = TRUE,
  merge_by_chunk_id,
  .progress = FALSE
)

## S3 method for class 'disk.frame'
inner_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE,
  outdir = tempfile("tmp_disk_frame_inner_join"),
  merge_by_chunk_id = NULL,
  overwrite = TRUE,
  .progress = FALSE
)

## S3 method for class 'disk.frame'
left_join(
  x,
  y,
  by = NULL,
  copy = FALSE,
  suffix = c(".x", ".y"),
  ...,
  keep = FALSE,
  outdir = tempfile("tmp_disk_frame_left_join"),
  merge_by_chunk_id = FALSE,
  overwrite = TRUE,
  .progress = FALSE
)

## S3 method for class 'disk.frame'
semi_join(
  x,
  y,
```

```

  by = NULL,
  copy = FALSE,
  ...,
  outdir = tempfile("tmp_disk_frame_semi_join"),
  merge_by_chunk_id = FALSE,
  overwrite = TRUE,
  .progress = FALSE
)

```

### Arguments

x	a disk.frame
y	a data.frame or disk.frame. If data.frame then returns lazily; if disk.frame it performs the join eagerly and return a disk.frame
by	join by
copy	same as dplyr::anti_join
...	same as dplyr's joins
outdir	output directory for disk.frame
merge_by_chunk_id	the merge is performed by chunk id
overwrite	overwrite output directory
.progress	Show progress or not. Defaults to FALSE
suffix	see dplyr::XXX_join
keep	see dplyr::XXX_join

### Value

disk.frame or data.frame/data.table

### Examples

```

df.df = as.disk.frame(data.frame(x = 1:3, y = 4:6), overwrite = TRUE)
df2.df = as.disk.frame(data.frame(x = 1:2, z = 10:11), overwrite = TRUE)

anti_joined.df = anti_join(df.df, df2.df)

anti_joined.df %>% collect

anti_joined.data.frame = anti_join(df.df, data.frame(x = 1:2, z = 10:11))

# clean up
delete(df.df)
delete(df2.df)
delete(anti_joined.df)
cars.df = as.disk.frame(cars)

join.df = full_join(cars.df, cars.df, merge_by_chunk_id = TRUE)

```

```

# clean up cars.df
delete(cars.df)
delete(join.df)
cars.df = as.disk.frame(cars)

join.df = inner_join(cars.df, cars.df, merge_by_chunk_id = TRUE)

# clean up cars.df
delete(cars.df)
delete(join.df)
cars.df = as.disk.frame(cars)

join.df = left_join(cars.df, cars.df)

# clean up cars.df
delete(cars.df)
delete(join.df)
cars.df = as.disk.frame(cars)

join.df = semi_join(cars.df, cars.df)

# clean up cars.df
delete(cars.df)
delete(join.df)

```

---

as.data.frame.disk.frame

*Convert disk.frame to data.frame by collecting all chunks*

---

## Description

Convert disk.frame to data.frame by collecting all chunks

## Usage

```

## S3 method for class 'disk.frame'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

```

## Arguments

x	a disk.frame
row.names	NULL or a character vector giving the row names for the data frame. Missing values are not allowed.
optional	logical. If TRUE, setting row names and converting column names (to syntactic names: see make.names) is optional. Note that all of R's base package as.data.frame() methods use optional only for column names treatment, basically with the meaning of data.frame(*, check.names = !optional). See also the make.names argument of the matrix method.
...	additional arguments to be passed to or from methods.

## Examples

```
cars.df = as.disk.frame(cars)
as.data.frame(cars.df)

# clean up
delete(cars.df)
```

---

```
as.data.table.disk.frame
```

*Convert disk.frame to data.table by collecting all chunks*

---

## Description

Convert disk.frame to data.table by collecting all chunks

## Usage

```
## S3 method for class 'disk.frame'
as.data.table(x, keep.rownames = FALSE, ...)
```

## Arguments

x	a disk.frame
keep.rownames	passed to as.data.table
...	passed to as.data.table

## Examples

```
library(data.table)
cars.df = as.disk.frame(cars)
as.data.table(cars.df)

# clean up
delete(cars.df)
```

---

```
as.disk.frame
```

*Make a data.frame into a disk.frame*

---

## Description

Make a data.frame into a disk.frame



**Usage**

```
as.disk.frame(  
  df,  
  outdir = tempfile(fileext = ".df"),  
  nchunks = recommend_nchunks(df),  
  overwrite = FALSE,  
  shardby = NULL,  
  compress = 50,  
  ...  
)
```

**Arguments**

df	a disk.frame
outdir	the output directory
nchunks	number of chunks
overwrite	if TRUE the outdir will be overwritten, if FALSE it will throw an error if the directory is not empty
shardby	The shardkey
compress	the compression level 0-100; 100 is highest
...	passed to output_disk.frame

**Examples**

```
# write to temporary location  
cars.df = as.disk.frame(cars)  
  
# specify a different path in the temporary folder, you are free to choose a different folder  
cars_new_location.df = as.disk.frame(cars, outdir = file.path(tempdir(), "some_path.df"))  
  
# specify a different number of chunks  
# this writes to tempdir() by default  
cars_chunks.df = as.disk.frame(cars, nchunks = 4, overwrite = TRUE)  
  
# clean up  
delete(cars.df)  
delete(cars_new_location.df)  
delete(cars_chunks.df)
```

---

bind\_rows.disk.frame *Bind rows*

---

**Description**

Bind rows

**Usage**

```
bind_rows.disk.frame(...)
```

**Arguments**

```
...          disk.frame to be row bound
```

---

```
chunk_summarize    #' @export #' @importFrom dplyr add_count #'
                   @rdname dplyr_verbs add_count.disk.frame <- cre-
                   ate_chunk_mapper(dplyr::add_count) #' @export #' @importFrom
                   dplyr add_tally #' @rdname dplyr_verbs add_tally.disk.frame <-
                   create_chunk_mapper(dplyr::add_tally)
```

---

**Description**

The `disk.frame` group by operation perform group WITHIN each chunk. This is often used for performance reasons. If the user wishes to perform group-by, they may choose to use the `'hard_group_by'` function which is expensive as it reorganizes the chunks by the shard key.

**Usage**

```
chunk_summarize(.data, ...)
```

```
chunk_summarise(.data, ...)
```

```
chunk_group_by(.data, ...)
```

```
chunk_ungroup(.data, ...)
```

**Arguments**

```
.data          a disk.frame
```

```
...           passed to dplyr::group_by
```

**See Also**

```
hard_group_by group_by
```

---

`cmap`*Apply the same function to all chunks*

---

**Description**

Apply the same function to all chunks

‘cmap.disk.frame’ accepts a two argument function where the first argument is a data.frame and the second is the chunk ID

‘lazy’ is convenience function to apply ‘f’ to every chunk

‘delayed’ is an alias for lazy and is consistent with the naming in Dask and Dagger.jl

**Usage**

```
cmap(.x, .f, ...)
```

```
## S3 method for class 'disk.frame'
```

```
cmap(.x, .f, ...)
```

```
cmap_dfr(.x, .f, ..., .id = NULL)
```

```
## S3 method for class 'disk.frame'
```

```
cmap_dfr(.x, .f, ..., .id = NULL, use.names = fill, fill = FALSE, idcol = NULL)
```

```
cimap(.x, .f, ...)
```

```
## S3 method for class 'disk.frame'
```

```
cimap(  
  .x,  
  .f,  
  outdir = NULL,  
  keep = NULL,  
  lazy = TRUE,  
  overwrite = FALSE,  
  compress = 50,  
  ...  
)
```

```
cimap_dfr(.x, .f, ..., .id = NULL)
```

```
## S3 method for class 'disk.frame'
```

```
cimap_dfr(  
  .x,  
  .f,  
  ...,  
  .id = NULL,  
  use.names = fill,
```

```

    fill = FALSE,
    idcol = NULL
  )

  lazy(.x, .f, ...)

  ## S3 method for class 'disk.frame'
  lazy(.x, .f, ...)

  delayed(.x, .f, ...)

  clapply(...)

```

### Arguments

<code>.x</code>	a <code>disk.frame</code>
<code>.f</code>	a function to apply to each of the chunks
<code>...</code>	Passed to <code>'collect'</code> and <code>'write_disk.frame'</code>
<code>.id</code>	ignored
<code>use.names</code>	for <code>cmap_dfr</code> 's call to <code>data.table::rbindlist</code> . See <code>data.table::rbindlist</code>
<code>fill</code>	for <code>cmap_dfr</code> 's call to <code>data.table::rbindlist</code> . See <code>data.table::rbindlist</code>
<code>idcol</code>	for <code>cmap_dfr</code> 's call to <code>data.table::rbindlist</code> . See <code>data.table::rbindlist</code>
<code>outdir</code>	the output directory
<code>keep</code>	The columns to keep at source
<code>lazy</code>	if <code>TRUE</code> then do this lazily
<code>overwrite</code>	Whether to overwrite any files in the output directory
<code>compress</code>	The compression setting. 0-100

### Examples

```

cars.df = as.disk.frame(cars)

# return the first row of each chunk lazily
#
cars2 = cmap(cars.df, function(chunk) {
  chunk[,1]
})

collect(cars2)

# same as above but using purrr
cars2 = cmap(cars.df, ~.x[1,])

collect(cars2)

# return the first row of each chunk eagerly as list
cmap(cars.df, ~.x[1,], lazy = FALSE)

```

```

# return the first row of each chunk eagerly as data.table/data.frame by row-binding
cmap_dfr(cars.df, ~.x[1,])

# lazy and delayed are just an aliases for cmap(..., lazy = TRUE)
collect(lazy(cars.df, ~.x[1,]))
collect(delayed(cars.df, ~.x[1,]))

# clean up cars.df
delete(cars.df)

```

---

cmap2

*'cmap2' a function to two disk.frames*


---

### Description

Perform a function on both disk.frames `.x` and `.y`, each chunk of `.x` and `.y` gets run by `.f(x.chunk, y.chunk)`

### Usage

```

cmap2(.x, .y, .f, ...)

map_by_chunk_id(.x, .y, .f, ..., outdir)

```

### Arguments

<code>.x</code>	a disk.frame
<code>.y</code>	a disk.frame
<code>.f</code>	a function to be called on each chunk of x and y matched by <code>chunk_id</code>
<code>...</code>	not used
<code>outdir</code>	output directory

### Examples

```

cars.df = as.disk.frame(cars)

cars2.df = cmap2(cars.df, cars.df, ~data.table::rbindlist(list(.x, .y)))
collect(cars2.df)

# clean up cars.df
delete(cars.df)
delete(cars2.df)

```

---

collect.disk.frame      *Bring the disk.frame into R*

---

### Description

Bring the disk.frame into RAM by loading the data and running all lazy operations as data.table/data.frame or as a list

Bring the disk.frame into RAM by loading the data and running all lazy operations as data.table/data.frame or as a list

### Usage

```
## S3 method for class 'disk.frame'
collect(x, ..., parallel = !is.null(attr(x, "recordings")))

collect_list(
  x,
  simplify = FALSE,
  parallel = !is.null(attr(x, "recordings")),
  ...
)

## S3 method for class 'summarized_disk.frame'
collect(x, ..., parallel = !is.null(attr(x, "recordings")))
```

### Arguments

x	a disk.frame
...	not used
parallel	if TRUE the collection is performed in parallel. By default if there are delayed/lazy steps then it will be parallel, otherwise it will not be in parallel. This is because parallel requires transferring data from background R session to the current R session and if there is no computation then it's better to avoid transferring data between session, hence parallel = FALSE is a better choice
simplify	Should the result be simplified to array

### Value

collect return a data.frame/data.table

collect\_list returns a list

collect return a data.frame/data.table

## Examples

```
cars.df = as.disk.frame(cars)
# use collect to bring the data into RAM as a data.table/data.frame
collect(cars.df)

# clean up
delete(cars.df)
cars.df = as.disk.frame(cars)

# returns the result as a list
collect_list(cmap(cars.df, ~1))

# clean up
delete(cars.df)
cars.df = as.disk.frame(cars)
# use collect to bring the data into RAM as a data.table/data.frame
collect(cars.df)

# clean up
delete(cars.df)
```

---

colnames

*Return the column names of the disk.frame*

---

## Description

The returned column names are from the source. So if you have lazy operations then the colnames here does not reflect the results of those operations. Note: if you have expensive lazy function then this operation might take some time.

## Usage

```
colnames(x, ...)
```

## S3 method for class 'disk.frame'

```
names(x, ...)
```

## S3 method for class 'disk.frame'

```
colnames(x, ...)
```

## Default S3 method:

```
colnames(x, ...)
```

## Arguments

x	a disk.frame
...	not used

---

compute.disk.frame      *Force computations. The results are stored in a folder.*

---

### Description

Perform the computation; same as calling `cmap` without `.f` and `lazy = FALSE`

### Usage

```
## S3 method for class 'disk.frame'
compute(x, name = NULL, outdir = tempfile("tmp_df_", fileext = ".df"), ...)
```

### Arguments

<code>x</code>	a <code>disk.frame</code>
<code>name</code>	If not <code>NULL</code> then used as <code>outdir</code> prefix.
<code>outdir</code>	the output directory
<code>...</code>	Passed to <code>'write_disk.frame'</code>

### Examples

```
cars.df = as.disk.frame(cars)
cars.df2 = cars.df %>% cmap(~.x)
# the computation is performed and the data is now stored elsewhere
cars.df3 = compute(cars.df2)

# clean up
delete(cars.df)
delete(cars.df3)
```

---

create\_chunk\_mapper      *Create function that applies to each chunk if disk.frame*

---

### Description

A function to make it easier to create functions like `filter`

### Usage

```
create_chunk_mapper(chunk_fn, warning_msg = NULL, as.data.frame = FALSE)
```

### Arguments

<code>chunk_fn</code>	The dplyr function to create a mapper for
<code>warning_msg</code>	The warning message to display when invoking the mapper
<code>as.data.frame</code>	force the input chunk of a <code>data.frame</code> ; needed for <code>dtplyr</code>



**Examples**

```

filter = create_chunk_mapper(dplyr::filter)

#' example: creating a function that keeps only the first and last n row
first_and_last <- function(chunk, n, ...) {
  nr = nrow(chunk)
  print(nr-n+1:nr)
  chunk[c(1:n, (nr-n+1):nr), ]
}

#' create the function for use with disk.frame
first_and_last_df = create_chunk_mapper(first_and_last)

mtcars.df = as.disk.frame(mtcars)

#' the operation is lazy
lazy_mtcars.df = mtcars.df %>%
  first_and_last_df(2)

#' bring into R
collect(lazy_mtcars.df)

#' clean up
delete(mtcars.df)

```

---

csv_to_disk.frame	<i>Convert CSV file(s) to disk.frame format</i>
-------------------	---

---

**Description**

Convert CSV file(s) to disk.frame format

**Usage**

```

csv_to_disk.frame(
  infile,
  outdir = tempfile(fileext = ".df"),
  inmapfn = base::I,
  nchunks = recommend_nchunks(sum(file.size(infile))),
  in_chunk_size = NULL,
  shardby = NULL,
  compress = 50,
  overwrite = TRUE,
  header = TRUE,
  .progress = TRUE,
  backend = c("data.table", "readr", "LaF"),

```

```

    chunk_reader = c("bigreadr", "data.table", "readr", "readLines"),
    ...
)

```

### Arguments

<code>infile</code>	The input CSV file or files
<code>outdir</code>	The directory to output the disk.frame to
<code>inmapfn</code>	A function to be applied to the chunk read in from CSV before the chunk is being written out. Commonly used to perform simple transformations. Defaults to the identity function (ie. no transformation)
<code>nchunks</code>	Number of chunks to output
<code>in_chunk_size</code>	When reading in the file, how many lines to read in at once. This is different to <code>nchunks</code> which controls how many chunks are output
<code>shardby</code>	The column(s) to shard the data by. For example suppose <code>'shardby = c("col1", "col2")'</code> then every row where the values <code>'col1'</code> and <code>'col2'</code> are the same will end up in the same chunk; this will allow merging by <code>'col1'</code> and <code>'col2'</code> to be more efficient
<code>compress</code>	For <code>fst</code> backends it's a number between 0 and 100 where 100 is the highest compression ratio.
<code>overwrite</code>	Whether to overwrite the existing directory
<code>header</code>	Whether the files have header. Defaults to TRUE
<code>.progress</code>	A logical, for whether or not to show progress
<code>backend</code>	The CSV reader backend to choose: "data.table" or "readr". <code>disk.frame</code> does not have its own CSV reader. It uses either <code>data.table::fread</code> or <code>readr::read_delimited</code> . It is worth noting that <code>data.table::fread</code> does not detect dates and all dates are imported as strings, and you are encouraged to use <code>fasttime</code> to convert the strings to date. You can use the <code>'inmapfn'</code> to do that. However, if you want automatic date detection, then <code>backend="readr"</code> may suit your needs. However, <code>readr</code> is often slower than <code>data.table</code> , hence <code>data.table</code> is chosen as the default.
<code>chunk_reader</code>	Even if you choose a backend there can still be multiple strategies on how to approach the CSV reads. For example, <code>data.table::fread</code> tries to <code>mmap</code> the whole file which can cause the whole read process to fail. In that case we can change the <code>chunk_reader</code> to "readLines" which uses the <code>readLines</code> function to read chunk by chunk and still use <code>data.table::fread</code> to process the chunks. There are currently no strategies for <code>readr</code> backend, except the default one.
<code>...</code>	passed to <code>data.table::fread</code> , <code>disk.frame::as.disk.frame</code> , <code>disk.frame::shard</code>

### See Also

Other ingesting data: [zip\\_to\\_disk.frame\(\)](#)

### Examples

```

tmpfile = tempfile()
write.csv(cars, tmpfile)
tmpdf = tempfile(fileext = ".df")

```

```
df = csv_to_disk.frame(tmpfile, outdir = tmpdf, overwrite = TRUE)

# clean up
fs::file_delete(tmpfile)
delete(df)
```

---

delete	<i>Delete a disk.frame</i>
--------	----------------------------

---

**Description**

Delete a disk.frame

**Usage**

```
delete(df)
```

**Arguments**

df                    a disk.frame

**Examples**

```
cars.df = as.disk.frame(cars)
delete(cars.df)
```

---

df_ram_size	<i>Get the size of RAM in gigabytes</i>
-------------	---

---

**Description**

Get the size of RAM in gigabytes

**Usage**

```
df_ram_size()
```

**Value**

integer of RAM in gigabyte (GB)

**Examples**

```
# returns the RAM size in gigabyte (GB)
df_ram_size()
```

---

disk.frame	<i>Create a disk.frame from a folder</i>
------------	--

---

**Description**

Create a disk.frame from a folder

**Usage**

```
disk.frame(path, backend = "fst")
```

**Arguments**

path	The path to store the output file or to a directory
backend	The only available backend is fst at the moment

**Examples**

```
path = file.path(tempdir(), "cars")
as.disk.frame(cars, outdir=path, overwrite = TRUE, nchunks = 2)
df = disk.frame(path)
head(df)
nchunks(df)
# clean up
delete(df)
```

---

disk.frame_to_parquet	<i>A function to convert a disk.frame to parquet format</i>
-----------------------	---

---

**Description**

A function to convert a disk.frame to parquet format

**Usage**

```
disk.frame_to_parquet(df, outdir)
```

**Arguments**

df	a disk.frame or a path to a disk.frame
outdir	the path to save the parquet files

---

evalparseglue	<i>Helper function to evalparse some 'glue::glue' string</i>
---------------	--

---

**Description**

Helper function to evalparse some 'glue::glue' string

**Usage**

```
evalparseglue(code, env = parent.frame())
```

**Arguments**

code	the code in character(string) format to evaluate
env	the environment in which to evaluate the code

---

find_globals_recursively	<i>Find globals in an expression by searching through the chain</i>
--------------------------	---

---

**Description**

Find globals in an expression by searching through the chain

**Usage**

```
find_globals_recursively(code, envir)
```

**Arguments**

code	An expression to search for globals
envir	The environment from which to begin the search

---

foverlaps.disk.frame *Apply data.table's foverlaps to the disk.frame*

---

## Description

EXPERIMENTAL

## Usage

```
foverlaps.disk.frame(
  df1,
  df2,
  by.x = if (identical(shardkey(df1)$shardkey, "")) shardkey(df1)$shardkey else
    shardkey(df2)$shardkey,
  by.y = shardkey(df2)$shardkey,
  ...,
  outdir = tempfile("df_foverlaps_tmp", fileext = ".df"),
  merge_by_chunk_id = FALSE,
  compress = 50,
  overwrite = TRUE
)
```

## Arguments

df1	A disk.frame
df2	A disk.frame or a data.frame
by.x	character/string vector. by.x used in foverlaps
by.y	character/string vector. by.x used in foverlaps
...	passed to data.table::foverlaps and disk.frame::cmap.disk.frame
outdir	The output directory of the disk.frame
merge_by_chunk_id	If TRUE then the merges will happen for chunks in df1 and df2 with the same chunk id which speed up processing. Otherwise every chunk of df1 is merged with every chunk of df2. Ignored with df2 is not a disk.frame
compress	The compression ratio for fst
overwrite	overwrite existing directory

## Examples

```
library(data.table)

## simple example:
x = as.disk.frame(data.table(start=c(5,31,22,16), end=c(8,50,25,18), val2 = 7:10))
y = as.disk.frame(data.table(start=c(10, 20, 30), end=c(15, 35, 45), val1 = 1:3))
byxy = c("start", "end")
```

```
xy.df = foverlaps.disk.frame(  
  x, y, by.x = byxy, by.y = byxy,  
  merge_by_chunk_id = TRUE, overwrite = TRUE)  
# clean up  
delete(x)  
delete(y)  
delete(xy.df)
```

---

gen\_datatable\_synthetic

*Generate synthetic dataset for testing*

---

### Description

Generate synthetic dataset for testing

### Usage

```
gen_datatable_synthetic(N = 2e+08, K = 100)
```

### Arguments

N	number of rows. Defaults to 200 million
K	controls the number of unique values for id. Some ids will have K distinct values while others have N/K distinct values

---

get\_chunk

*Obtain one chunk by chunk id*

---

### Description

Obtain one chunk by chunk id

### Usage

```
get_chunk(...)
```

```
## S3 method for class 'disk.frame'
```

```
get_chunk(df, n, keep = NULL, full.names = FALSE, ..., partitioned_info = NULL)
```

**Arguments**

...	passed to fst::read_fst or whichever read function is used in the backend
df	a disk.frame
n	the chunk id. If numeric then matches by number, if character then returns the chunk with the same name as n
keep	the columns to keep
full.names	whether n is the full path to the chunks or just a relative path file name. Ignored if n is numeric
partitioned_info	for internal use only. It's a data.frame used to help with filtering by partitions

**Examples**

```
cars.df = as.disk.frame(cars, nchunks = 2)
get_chunk(cars.df, 1)
get_chunk(cars.df, 2)
get_chunk(cars.df, 1, keep = "speed")

# if full.names = TRUE then the full path to the chunk need to be provided
get_chunk(cars.df, file.path(attr(cars.df, "path"), "1.fst"), full.names = TRUE)

# clean up cars.df
delete(cars.df)
```

---

get_chunk_ids	<i>Get the chunk IDs and files names</i>
---------------	--

---

**Description**

Get the chunk IDs and files names

**Usage**

```
get_chunk_ids(df, ..., full.names = FALSE, strip_extension = TRUE)
```

**Arguments**

df	a disk.frame
...	passed to list.files
full.names	If TRUE returns the full path to the file, Defaults to FALSE
strip_extension	If TRUE then the file extension in the chunk_id is removed. Defaults to TRUE



**Examples**

```
cars.df = as.disk.frame(cars)

# return the integer-string chunk IDs
get_chunk_ids(cars.df)

# return the file name chunk IDs
get_chunk_ids(cars.df, full.names = TRUE)

# return the file name chunk IDs with file extension
get_chunk_ids(cars.df, strip_extension = FALSE)

# clean up cars.df
delete(cars.df)
```

---

get\_partition\_paths     *Get the partitioning structure of a folder*

---

**Description**

Get the partitioning structure of a folder

**Usage**

```
get_partition_paths(df)
```

**Arguments**

df                    a disk.frame whose paths will be used to determine if it's folder-partitioned disk.frame

---

groups.disk.frame     *The shard keys of the disk.frame*

---

**Description**

The shard keys of the disk.frame

**Usage**

```
## S3 method for class 'disk.frame'
groups(x)
```

**Arguments**

x                    a disk.frame

**Value**

character

---

head.disk.frame	<i>Head and tail of the disk.frame</i>
-----------------	--

---

**Description**

Head and tail of the disk.frame

**Usage**

```
## S3 method for class 'disk.frame'
head(x, n = 6L, ...)

## S3 method for class 'disk.frame'
tail(x, n = 6L, ...)
```

**Arguments**

x	a disk.frame
n	number of rows to include
...	passed to base::head or base::tail

**Examples**

```
cars.df = as.disk.frame(cars)
head(cars.df)
tail(cars.df)

# clean up
delete(cars.df)
```

---

is_disk.frame	<i>Checks if a folder is a disk.frame</i>
---------------	---

---

**Description**

Checks if a folder is a disk.frame

**Usage**

```
is_disk.frame(df)
```

**Arguments**

df                    a disk.frame or directory to check

**Examples**

```
cars.df = as.disk.frame(cars)

is_disk.frame(cars) # FALSE
is_disk.frame(cars.df) # TRUE

# clean up cars.df
delete(cars.df)
```

---

merge.disk.frame            *Merge function for disk.frames*

---

**Description**

Merge function for disk.frames

**Usage**

```
## S3 method for class 'disk.frame'
merge(
  x,
  y,
  by,
  outdir = tempfile(fileext = ".df"),
  ...,
  merge_by_chunk_id = FALSE,
  overwrite = FALSE
)
```

**Arguments**

x                    a disk.frame

y                    a disk.frame or data.frame

by                    the merge by keys

outdir                The output directory for the disk.frame

...                    passed to merge and cmap.disk.frame

merge\_by\_chunk\_id    if TRUE then only chunks in df1 and df2 with the same chunk id will get merged

overwrite             overwrite the outdir or not

## Examples

```
b = as.disk.frame(data.frame(a = 51:150, b = 1:100))
d = as.disk.frame(data.frame(a = 151:250, b = 1:100))
bd.df = merge(b, d, by = "b", merge_by_chunk_id = TRUE)

# clean up cars.df
delete(b)
delete(d)
delete(bd.df)
```

---

move\_to

*Move or copy a disk.frame to another location*

---

## Description

Move or copy a disk.frame to another location

## Usage

```
move_to(df, outdir, ..., copy = FALSE)
```

```
copy_df_to(df, outdir, ...)
```

## Arguments

df	The disk.frame
outdir	The new location
...	NOT USED
copy	Merely copy and not move

## Value

a disk.frame

## Examples

```
cars.df = as.disk.frame(cars)

cars_copy.df = copy_df_to(cars.df, outdir = tempfile(fileext=".df"))

cars2.df = move_to(cars.df, outdir = tempfile(fileext=".df"))

# clean up
delete(cars_copy.df)
delete(cars2.df)
```

---

nchunks	<i>Returns the number of chunks in a disk.frame</i>
---------	---

---

### Description

Returns the number of chunks in a disk.frame

### Usage

```
nchunks(df, ...)  
  
nchunk(df, ...)  
  
## S3 method for class 'disk.frame'  
nchunk(df, ...)  
  
## S3 method for class 'disk.frame'  
nchunks(df, skip.ready.check = FALSE, ...)
```

### Arguments

df	a disk.frame
...	not used
skip.ready.check	NOT implemented

### Examples

```
cars.df = as.disk.frame(cars)  
  
# return the number of chunks  
nchunks(cars.df)  
nchunk(cars.df)  
  
# clean up cars.df  
delete(cars.df)
```

---

nrow	<i>Number of rows or columns</i>
------	----------------------------------

---

### Description

Number of rows or columns

**Usage**

```
nrow(df, ...)

## S3 method for class 'disk.frame'
nrow(df, ...)

ncol(df)

## S3 method for class 'disk.frame'
ncol(df)
```

**Arguments**

```
df          a disk.frame
...         passed to base::nrow
```

**Examples**

```
cars.df = as.disk.frame(cars)

# return total number of column and rows
ncol(cars.df)
nrow(cars.df)

# clean up cars.df
delete(cars.df)
```

---

```
overwrite_check      Check if the outdir exists or not
```

---

**Description**

If the overwrite is TRUE then the folder will be deleted, otherwise the folder will be created.

**Usage**

```
overwrite_check(outdir, overwrite)
```

**Arguments**

```
outdir      the output directory
overwrite   TRUE or FALSE if 'outdir' exists and overwrite = FALSE then throw an error
```

**Examples**

```
tf = tempfile()
overwrite_check(tf, overwrite = FALSE)
overwrite_check(tf, overwrite = TRUE)

# clean up
fs::dir_delete(tf)
```

---

partition_filter	<i>Filter the dataset based on folder partitions</i>
------------------	--

---

**Description**

Filter the dataset based on folder partitions

**Usage**

```
partition_filter(x, ...)
```

**Arguments**

x	a disk.frame
...	filtering conditions for filtering the disk.frame at (folder) partition level

---

play	<i>Play the recorded lazy operations</i>
------	--

---

**Description**

Play the recorded lazy operations

**Usage**

```
play(dataframe, recordings)
```

**Arguments**

dataframe	A data.frame
recordings	A recording the expression, globals and packages using create_chunk_mapper

---

print.disk.frame	<i>Print disk.frame</i>
------------------	-------------------------

---

**Description**

a new print method for disk.frame

**Usage**

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

**Arguments**

x	disk.frame
...	not used

---

pull.disk.frame	<i>Pull a column from table similar to 'dplyr::pull'.</i>
-----------------	---

---

**Description**

Pull a column from table similar to 'dplyr::pull'.

**Usage**

```
## S3 method for class 'disk.frame'
pull(.data, var = -1, name = NULL, ...)
```

**Arguments**

.data	The disk.frame
var	can be an positive or negative integer or a character/string. See dplyr::pull documentation
name	See dplyr::pull documentation
...	Not used, kept for compatibility with 'dplyr::pull'



---

purrr_as_mapper	<i>Used to convert a function to purrr syntax if needed</i>
-----------------	---

---

**Description**

Used to convert a function to purrr syntax if needed

**Usage**

```
purrr_as_mapper(.f)
```

**Arguments**

.f	a normal function or purrr syntax function i.e. ‘~ ...code...’
----	--

---

rbindlist.disk.frame	<i>rbindlist disk.frames together</i>
----------------------	---------------------------------------

---

**Description**

rbindlist disk.frames together

**Usage**

```
rbindlist.disk.frame(
  df_list,
  outdir = tempfile(fileext = ".df"),
  by_chunk_id = TRUE,
  parallel = TRUE,
  compress = 50,
  overwrite = TRUE,
  .progress = TRUE
)
```

**Arguments**

df_list	A list of disk.frames
outdir	Output directory of the row-bound disk.frames
by_chunk_id	If TRUE then only the chunks with the same chunk IDs will be bound
parallel	if TRUE then bind multiple disk.frame simultaneously, Defaults to TRUE
compress	0-100, 100 being the highest compression rate.
overwrite	overwrite the output directory
.progress	A logical, for whether or not to show progress.

## Examples

```
cars.df = as.disk.frame(cars)

# row-bind two disk.frames
cars2.df = rbindlist.disk.frame(list(cars.df, cars.df))

# clean up cars.df
delete(cars.df)
delete(cars2.df)
```

---

rechunk

*Increase or decrease the number of chunks in the disk.frame*

---

## Description

Increase or decrease the number of chunks in the disk.frame

## Usage

```
rechunk(
  df,
  nchunks = disk.frame::nchunks(df),
  outdir = attr(df, "path", exact = TRUE),
  shardby = NULL,
  overwrite = TRUE
)
```

## Arguments

df	the disk.frame to rechunk
nchunks	number of chunks
outdir	the output directory
shardby	the shardkeys
overwrite	overwrite the output directory

## Examples

```
# create a disk.frame with 2 chunks in tempdir()
cars.df = as.disk.frame(cars, nchunks = 2)

# re-chunking cars.df to 3 chunks, done "in-place" to the same folder as cars.df
rechunk(cars.df, 3)

new_path = tempfile(fileext = ".df")
# re-chunking cars.df to 4 chunks, shard by speed, and done "out-of-place" to a new directory
cars2.df = rechunk(cars.df, 4, outdir=new_path, shardby = "speed")
```

```
# clean up cars.df
delete(cars.df)
delete(cars2.df)
```

---

recommend_nchunks	<i>Recommend number of chunks based on input size</i>
-------------------	---

---

### Description

Computes the recommended number of chunks to break a data.frame into. It can accept filesizes in bytes (as integer) or a data.frame

### Usage

```
recommend_nchunks(
  df,
  type = "csv",
  minchunks = data.table::getDTthreads(),
  conservatism = 8,
  ram_size = df_ram_size()
)
```

### Arguments

df	a disk.frame or the file size in bytes of a CSV file holding the data
type	only = "csv" is supported. It indicates the file type corresponding to file size 'df'
minchunks	the minimum number of chunks. Defaults to the number of CPU cores (without hyper-threading)
conservatism	a multiplier to the recommended number of chunks. The more chunks the smaller the chunk size and more likely that each chunk can fit into RAM
ram_size	The amount of RAM available which is usually computed. Except on RStudio with R3.6+

### Examples

```
# recommend nchunks based on data.frame
recommend_nchunks(cars)

# recommend nchunks based on file size ONLY CSV is implemented at the moment
recommend_nchunks(1024^3)
```

---

remove_chunk	<i>Removes a chunk from the disk.frame</i>
--------------	--

---

**Description**

Removes a chunk from the disk.frame

**Usage**

```
remove_chunk(df, chunk_id, full.names = FALSE)
```

**Arguments**

df	a disk.frame
chunk_id	the chunk ID of the chunk to remove. If it's a number then return number.fst
full.names	TRUE or FALSE. Defaults to FALSE. If true then chunk_id is the full path to the chunk otherwise it's the relative path

**Examples**

```
# TODO add these to tests
cars.df = as.disk.frame(cars, nchunks = 4)

# removes 3rd chunk
remove_chunk(cars.df, 3)
nchunks(cars.df) # 3

# removes 4th chunk
remove_chunk(cars.df, "4.fst")
nchunks(cars.df) # 3

# removes 2nd chunk
remove_chunk(cars.df, file.path(attr(cars.df, "path", exact=TRUE), "2.fst"), full.names = TRUE)
nchunks(cars.df) # 1

# clean up cars.df
delete(cars.df)
```

---

sample_frac.disk.frame
------------------------

*Sample n rows from a disk.frame*

---

**Description**

Sample n rows from a disk.frame

**Usage**

```
## S3 method for class 'disk.frame'
sample_frac(tbl, size = 1, replace = FALSE, weight = NULL, .env = NULL, ...)
```

**Arguments**

tbl	A data.frame.
size	<tidy-select> For sample_n(), the number of rows to select. For sample_frac(), the fraction of rows to select. If tbl is grouped, size applies to each group.
replace	Sample with or without replacement?
weight	<tidy-select> Sampling weights. This must evaluate to a vector of non-negative numbers the same length as the input. Weights are automatically standardised to sum to 1.
.env	DEPRECATED.
...	ignored

**Examples**

```
cars.df = as.disk.frame(cars)

collect(sample_frac(cars.df, 0.5))

# clean up cars.df
delete(cars.df)
```

---

select.disk.frame      *The dplyr verbs implemented for disk.frame*

---

**Description**

Please see the dplyr document for their usage. Please note ‘chunk\_arrange’ performs the actions within each chunk

**Usage**

```
## S3 method for class 'disk.frame'
select(.data, ...)

## S3 method for class 'disk.frame'
rename(.data, ...)

## S3 method for class 'disk.frame'
filter(.data, ...)

## S3 method for class 'disk.frame'
mutate(.data, ...)
```

```
## S3 method for class 'disk.frame'  
transmute(.data, ...)  
  
## S3 method for class 'disk.frame'  
arrange(.data, ...)  
  
chunk_arrange(.data, ...)  
  
## S3 method for class 'disk.frame'  
distinct(...)  
  
chunk_distinct(.data, ...)  
  
## S3 method for class 'disk.frame'  
glimpse(x, width = NULL, ...)
```

### Arguments

.data	a disk.frame
...	Same as the dplyr functions
x	'dplyr::glimpse' parameter
width	'dplyr::glimpse' parameter

### Examples

```
library(dplyr)  
cars.df = as.disk.frame(cars)  
mult = 2  
  
# use all any of the supported dplyr  
cars2 = cars.df %>%  
  select(speed) %>%  
  mutate(speed2 = speed * mult) %>%  
  filter(speed < 50) %>%  
  rename(speed1 = speed) %>%  
  collect  
  
# clean up cars.df  
delete(cars.df)
```

---

setup_disk.frame	<i>Set up disk.frame environment</i>
------------------	--------------------------------------

---

### Description

Set up disk.frame environment

**Usage**

```
setup_disk.frame(
  workers = data.table::getDTthreads(),
  future_backend = future::multisession,
  ...,
  gui = FALSE
)
```

**Arguments**

workers	the number of workers (background R processes in the
future_backend	which future backend to use for parallelization
...	passed to ‘future::plan‘
gui	Whether to use a Graphical User Interface (GUI) for selecting the options. Defaults to FALSE

**Examples**

```
if (interactive()) {
  # setup disk.frame to use multiple workers these may use more than two
  # cores, and is therefore not allowed on CRAN. Hence it's set to run only in
  # interactive session
  setup_disk.frame()

  # use a Shiny GUI to adjust settings
  # only run in interactive()
  setup_disk.frame(gui = TRUE)
}

# set the number workers to 2
setup_disk.frame(2)

# if you do not wish to use multiple workers you can set it to sequential
setup_disk.frame(future_backend=future::sequential)
```

---

shard	<i>Shard a data.frame/data.table or disk.frame into chunk and saves it into a disk.frame</i>
-------	--

---

**Description**

Shard a data.frame/data.table or disk.frame into chunk and saves it into a disk.frame

‘distribute‘ is an alias for ‘shard‘

**Usage**

```
shard(
  df,
  shardby,
  outdir = tempfile(fileext = ".df"),
  ...,
  nchunks = recommend_nchunks(df),
  overwrite = FALSE
)

distribute(...)
```

**Arguments**

df	A data.frame/data.table or disk.frame. If disk.frame, then <code>rechunk(df, ...)</code> is run
shardby	The column(s) to shard the data by.
outdir	The output directory of the disk.frame
...	not used
nchunks	The number of chunks
overwrite	If TRUE then the chunks are overwritten

**Examples**

```
# shard the cars data.frame by speed so that rows with the same speed are in the same chunk
iris.df = shard(iris, "Species")

# clean up cars.df
delete(iris.df)
```

---

shardkey	<i>Returns the shardkey (not implemented yet)</i>
----------	---

---

**Description**

Returns the shardkey (not implemented yet)

**Usage**

```
shardkey(df)
```

**Arguments**

df	a disk.frame
----	--------------



---

shardkey_equal	<i>Compare two disk.frame shardkeys</i>
----------------	---

---

**Description**

Compare two disk.frame shardkeys

**Usage**

```
shardkey_equal(sk1, sk2)
```

**Arguments**

sk1	shardkey1
sk2	shardkey2

---

show_ceremony	<i>Show the code to setup disk.frame</i>
---------------	--

---

**Description**

Show the code to setup disk.frame

**Usage**

```
show_ceremony()
ceremony_text()
show_boilerplate()
insert_ceremony()
```

---

split_string_into_df	<i>Turn a string of the form /partition1=val/partion2=val2 into data.frame</i>
----------------------	--

---

**Description**

Turn a string of the form /partition1=val/partion2=val2 into data.frame

**Usage**

```
split_string_into_df(path_strs)
```

**Arguments**

path_strs	The paths in string form to break into partition format
-----------	---

---

srckeeper	<i>Keep only the variables from the input listed in selections</i>
-----------	--

---

**Description**

Keep only the variables from the input listed in selections

**Usage**

```
srckeeper(diskf, selections, ...)
```

**Arguments**

diskf	a disk.frame
selections	The list of variables to keep from the input source
...	not yet used

**Examples**

```
cars.df = as.disk.frame(cars)

# when loading cars's chunks into RAM, load only the column speed
collect(srckeeper(cars.df, "speed"))

# clean up cars.df
delete(cars.df)
```

---

summarise.grouped_disk.frame	<i>A function to parse the summarize function</i>
------------------------------	---

---

**Description**

The disk.frame group by operation perform group WITHIN each chunk. This is often used for performance reasons. If the user wishes to perform group-by, they may choose to use the ‘hard\_group\_by’ function which is expensive as it reorganizes the chunks by the shard key.

**Usage**

```
## S3 method for class 'grouped_disk.frame'
summarise(.data, ...)

## S3 method for class 'grouped_disk.frame'
summarize(.data, ...)
```

```
## S3 method for class 'disk.frame'
group_by(
  .data,
  ...,
  .add = FALSE,
  .drop = stop("disk.frame does not support `.drop` in `group_by` at this stage")
)

## S3 method for class 'disk.frame'
summarize(.data, ...)

## S3 method for class 'disk.frame'
summarise(.data, ...)
```

### Arguments

.data	a disk.frame
...	same as the dplyr::group_by
.add	from dplyr
.drop	from dplyr

### See Also

hard\_group\_by

---

tbl\_vars.disk.frame    *Column names for RStudio auto-complete*

---

### Description

Returns the names of the columns. Needed for RStudio to complete variable names

### Usage

```
## S3 method for class 'disk.frame'
tbl_vars(x)

## S3 method for class 'disk.frame'
group_vars(x)
```

### Arguments

x	a disk.frame
---	--------------

---

var\_df.chunk\_agg.disk.frame

*One Stage function*

---

## Description

One Stage function  
mean chunk\_agg  
mean collected\_agg

## Usage

```
var_df.chunk_agg.disk.frame(x, na.rm = FALSE)
var_df.collected_agg.disk.frame(listx)
sd_df.chunk_agg.disk.frame(x, na.rm = FALSE)
sd_df.collected_agg.disk.frame(listx)
mean_df.chunk_agg.disk.frame(x, na.rm = FALSE, ...)
mean_df.collected_agg.disk.frame(listx)
sum_df.chunk_agg.disk.frame(x, ...)
sum_df.collected_agg.disk.frame(listx, ...)
min_df.chunk_agg.disk.frame(x, ...)
min_df.collected_agg.disk.frame(listx, ...)
max_df.chunk_agg.disk.frame(x, ...)
max_df.collected_agg.disk.frame(listx, ...)
median_df.chunk_agg.disk.frame(x, ...)
median_df.collected_agg.disk.frame(listx, ...)
n_df.chunk_agg.disk.frame(...)
n_df.collected_agg.disk.frame(listx, ...)
length_df.chunk_agg.disk.frame(x, ...)
```

```

length_df.collected_agg.disk.frame(listx, ...)
any_df.chunk_agg.disk.frame(x, ...)
any_df.collected_agg.disk.frame(listx, ...)
all_df.chunk_agg.disk.frame(x, ...)
all_df.collected_agg.disk.frame(listx, ...)
n_distinct_df.chunk_agg.disk.frame(x, na.rm = FALSE, ...)
n_distinct_df.collected_agg.disk.frame(listx, ...)
quantile_df.chunk_agg.disk.frame(x, ...)
quantile_df.collected_agg.disk.frame(listx, ...)
IQR_df.chunk_agg.disk.frame(x, na.rm = FALSE, ...)
IQR_df.collected_agg.disk.frame(listx, ...)

```

### Arguments

x	the input
na.rm	Remove NAs. TRUE or FALSE
listx	a list
...	additional options

---

write_disk.frame	<i>Write disk.frame to disk</i>
------------------	---------------------------------

---

### Description

Write a data.frame/disk.frame to a disk.frame location. If df is a data.frame then using the as.disk.frame function is recommended for most cases

### Usage

```

write_disk.frame(
  diskf,
  outdir = tempfile(fileext = ".df"),
  nchunks = ifelse("disk.frame" %in% class(diskf), nchunks.disk.frame(diskf),
    recommend_nchunks(diskf)),
  overwrite = FALSE,
  shardby = NULL,

```

```

    partitionby = NULL,
    compress = 50,
    ...
)

output_disk.frame(...)

```

### Arguments

diskf	a disk.frame
outdir	output directory for the disk.frame
nchunks	number of chunks
overwrite	overwrite output directory
shardby	the columns to shard by
partitionby	the columns to (folder) partition by
compress	compression ratio for fst files
...	passed to cmap.disk.frame

### Examples

```

cars.df = as.disk.frame(cars)

# write out a lazy disk.frame to disk
cars2.df = write_disk.frame(cmap(cars.df, ~.x[1,]), overwrite = TRUE)
collect(cars2.df)

# clean up cars.df
delete(cars.df)
delete(cars2.df)

```

---

zip_to_disk.frame	<i>'zip_to_disk.frame' is used to read and convert every CSV file within the zip file to disk.frame format</i>
-------------------	--

---

### Description

'zip\_to\_disk.frame' is used to read and convert every CSV file within the zip file to disk.frame format

### Usage

```

zip_to_disk.frame(
  zipfile,
  outdir,
  ...,
  validation.check = FALSE,
  overwrite = TRUE
)

```

**Arguments**

zipfile	The zipfile
outdir	The output directory for disk.frame
...	passed to fread
validation.check	should the function perform a check at the end to check for validity of output. It can detect issues with conversion
overwrite	overwrite output directory

**Value**

a list of disk.frame

**See Also**

Other ingesting data: [csv\\_to\\_disk.frame\(\)](#)

**Examples**

```
# create a zip file containing a csv
csvfile = tempfile(fileext = ".csv")
write.csv(cars, csvfile)
zipfile = tempfile(fileext = ".zip")
zip(zipfile, csvfile)

# read every file and convert it to a disk.frame
zip.df = zip_to_disk.frame(zipfile, tempfile(fileext = ".df"))

# there is only one csv file so it return a list of one disk.frame
zip.df[[1]]

# clean up
unlink(csvfile)
unlink(zipfile)
delete(zip.df[[1]])
```

---

[[.disk.frame

[[ interface for disk.frame using fst backend

---

**Description**

[[ interface for disk.frame using fst backend

**Usage**

```
## S3 method for class 'disk.frame'  
  
df[[  
  ...,  
  keep = NULL,  
  rbind = TRUE,  
  use.names = TRUE,  
  fill = FALSE,  
  idcol = NULL  
]]
```

**Arguments**

df	a disk.frame
...	same as data.table
keep	the columns to srckep
rbind	Whether to rbind the chunks. Defaults to TRUE
use.names	Same as in data.table::rbindlist
fill	Same as in data.table::rbindlist
idcol	Same as in data.table::rbindlist

**Examples**

```
cars.df = as.disk.frame(cars)  
speed_limit = 50  
cars.df[[speed < speed_limit ,.N, cut(dist, pretty(dist))]]  
  
# clean up  
delete(cars.df)
```



# Index

- \* **dplyr verbs**
  - select.disk.frame, 37
- \* **ingesting data**
  - csv\_to\_disk.frame, 17
  - zip\_to\_disk.frame, 46
- [[.disk.frame, 47
- add\_chunk, 3
- all\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), 44
- all\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), 44
- anti\_join.disk.frame, 4
- any\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), 44
- any\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), 44
- arrange.disk.frame (select.disk.frame), 37
- as.data.frame.disk.frame, 7
- as.data.table.disk.frame, 8
- as.disk.frame, 8
- bind\_rows.disk.frame, 9
- ceremony\_text (show\_ceremony), 41
- chunk\_arrange (select.disk.frame), 37
- chunk\_distinct (select.disk.frame), 37
- chunk\_group\_by (chunk\_summarize), 10
- chunk\_summarise (chunk\_summarize), 10
- chunk\_summarize, 10
- chunk\_ungroup (chunk\_summarize), 10
- cimap (cmap), 11
- cimap\_dfr (cmap), 11
- clapply (cmap), 11
- cmap, 11
- cmap2, 13
- cmap\_dfr (cmap), 11
- collect.disk.frame, 14
- collect.summarized\_disk.frame
  - (collect.disk.frame), 14
- collect\_list (collect.disk.frame), 14
- colnames, 15
- compute.disk.frame, 16
- copy\_df\_to (move\_to), 28
- create\_chunk\_mapper, 16
- csv\_to\_disk.frame, 17, 47
- delayed (cmap), 11
- delete, 19
- df\_ram\_size, 19
- disk.frame, 20
- disk.frame\_to\_parquet, 20
- distinct.disk.frame
  - (select.disk.frame), 37
- distribute (shard), 39
- evalparseglue, 21
- filter.disk.frame (select.disk.frame), 37
- find\_globals\_recursively, 21
- foverlaps.disk.frame, 22
- full\_join.disk.frame
  - (anti\_join.disk.frame), 4
- gen\_datatable\_synthetic, 23
- get\_chunk, 23
- get\_chunk\_ids, 24
- get\_partition\_paths, 25
- glimpse.disk.frame (select.disk.frame), 37
- group\_by.disk.frame
  - (summarise.grouped\_disk.frame), 42
- group\_vars.disk.frame
  - (tbl\_vars.disk.frame), 43

- groups.disk.frame, [25](#)
- head.disk.frame, [26](#)
- inner\_join.disk.frame
  - (anti\_join.disk.frame), [4](#)
- insert\_ceremony(show\_ceremony), [41](#)
- IQR\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- IQR\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- is\_disk.frame, [26](#)
- lazy(cmap), [11](#)
- left\_join.disk.frame
  - (anti\_join.disk.frame), [4](#)
- length\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- length\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- map\_by\_chunk\_id(cmap2), [13](#)
- max\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- max\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- mean\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- mean\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- median\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- median\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- merge.disk.frame, [27](#)
- min\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- min\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- move\_to, [28](#)
- mutate.disk.frame(select.disk.frame), [37](#)
- n\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- n\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- n\_distinct\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- n\_distinct\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- names.disk.frame(colnames), [15](#)
- nchunk(nchunks), [29](#)
- nchunks, [29](#)
- ncol(nrow), [29](#)
- nrow, [29](#)
- output\_disk.frame(write\_disk.frame), [45](#)
- overwrite\_check, [30](#)
- partition\_filter, [31](#)
- play, [31](#)
- print.disk.frame, [32](#)
- pull.disk.frame, [32](#)
- purrr\_as\_mapper, [33](#)
- quantile\_df.chunk\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- quantile\_df.collected\_agg.disk.frame
  - (var\_df.chunk\_agg.disk.frame), [44](#)
- rbindlist.disk.frame, [33](#)
- rechunk, [34](#)
- recommend\_nchunks, [35](#)
- remove\_chunk, [36](#)
- rename.disk.frame(select.disk.frame), [37](#)
- sample\_frac.disk.frame, [36](#)

sd\_df.chunk\_agg.disk.frame  
    (var\_df.chunk\_agg.disk.frame),  
    44

sd\_df.collected\_agg.disk.frame  
    (var\_df.chunk\_agg.disk.frame),  
    44

select.disk.frame, 37

semi\_join.disk.frame  
    (anti\_join.disk.frame), 4

setup\_disk.frame, 38

shard, 39

shardkey, 40

shardkey\_equal, 41

show\_boilerplate (show\_ceremony), 41

show\_ceremony, 41

split\_string\_into\_df, 41

srckeeper, 42

sum\_df.chunk\_agg.disk.frame  
    (var\_df.chunk\_agg.disk.frame),  
    44

sum\_df.collected\_agg.disk.frame  
    (var\_df.chunk\_agg.disk.frame),  
    44

summarise.disk.frame  
    (summarise.grouped\_disk.frame),  
    42

summarise.grouped\_disk.frame, 42

summarize.disk.frame  
    (summarise.grouped\_disk.frame),  
    42

summarize.grouped\_disk.frame  
    (summarise.grouped\_disk.frame),  
    42

tail.disk.frame (head.disk.frame), 26

tbl\_vars.disk.frame, 43

transmute.disk.frame  
    (select.disk.frame), 37

var\_df.chunk\_agg.disk.frame, 44

var\_df.collected\_agg.disk.frame  
    (var\_df.chunk\_agg.disk.frame),  
    44

write\_disk.frame, 45

zip\_to\_disk.frame, 18, 46