Package 'RMVL'

December 4, 2023

Title Mappable Vector Library for Handling Large Datasets

Version 1.0.0.1

Description Mappable vector library provides convenient way to access large datasets. Use all of your data at once, with few limits. Memory mapped data can be shared between multiple R processes. Access speed depends on storage medium, so solid state drive is recommended, preferably with PCI Express (or M.2 nvme) interface or a fast network file system. The data is memory mapped into R and then accessed using usual R list and array subscription operators. Convenience functions are provided for merging, grouping and indexing large vectors and data.frames. The layout of underlying MVL files is optimized for large datasets. The vectors are stored to guarantee alignment for vector intrinsics after memory map. The package is built on top of libMVL, which can be used as a standalone C library. libMVL has simple C API making it easy to interchange datasets with outside programs. Large MVL datasets are distributed via Academic Torrents <https://academictorrents.com/collection/mvl-datasets>.

URL https://academictorrents.com/collection/mvl-datasets, https://github.com/volodya31415/RMVL, https://github.com/volodya31415/libMVL

License LGPL-2.1

Depends R (>= 3.5.0)

Encoding UTF-8

RoxygenNote 7.2.3

NeedsCompilation yes

Author Vladimir Dergachev [aut, cre] (<https://orcid.org/0000-0003-4708-6625>)

Maintainer Vladimir Dergachev <support@altumrete.com>

Repository CRAN

Date/Publication 2023-12-04 16:10:02 UTC

R topics documented:

dim.MVL_OBJECT *Obtain dimensions of MVL object*

Description

Obtain dimensions of MVL object

Usage

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

Arguments

x	MVL_OBJECT as retrieved by subscription operators

Value

object dimensions, or NULL if not present

length.MVL_OBJECT *Obtain length of MVL object*

Description

Obtain length of MVL object

Usage

```
## S3 method for class 'MVL_OBJECT'
length(x)
```

Arguments

x	MVL_OBJECT as retrieved by subscription operators

Value

object length as stored in MVL file. This is the total length of object for arrays, and number of columns for data frames.

mvl2R *Make sure the object is fully converted to its R representation*

Description

If the object is stored in MVL file, we return its pure R representation. Otherwise, we return the object itself.

Usage

```
mvl2R(obj, raw = FALSE)
```

Arguments

obj	- MVL object retrieved by subscription of MVL library or other objects
raw	- request to return data in raw format when it does not map exactly to R data types.

Value

Stored object

mvl_add_directory_entries *Add entries to MVL directory*

Description

Add one or more entries to MVL directory

Usage

```
mvl_add_directory_entries(MVLHANDLE, tag, offsets)
```

Arguments

MVLHANDLE	handle to open MVL file created by mvl_open
tag	a vector of one or more character tags
offsets	a vector of MVL_OFFSET objects, one per tag, created by mvl_write_object

Details

This function is used to expand MVL directory. The offsets must be created by calling mvl_write_object on the same handle. Note that mvl_write_object has an optional parameter name that will add an entry when specified. Thus this function is meant for special circumstances, such as creating multiple entries in the directory that point to the same offset

mvl_class *Return underlying R class of object*

Description

This function returns the equivalent R class of underlying MVL object, i.e. the class it would have if converted into a regular R object. For non-MVL objects the function simply calls the usual R class(), so it can be used instead of class() for code that operates on both usual R objects and MVL objects.

Usage

```
mvl_class(x)
```

Arguments

x	any object

Value

character string giving object class

mvl_close *Close MVL file*

Description

Closes MVL file releasing all resources. For read-only files the memory is unmapped, reducing the virtual memory footprint. For files opened for writing the directory is written out, so it is important to call `mvl_close` or the newly written file will be corrupt. After `mvl_close()` all previously obtained MVL_OBJECT's with this handle become invalid.

Usage

```
mvl_close(MVLHANDLE)
```

Arguments

MVLHANDLE	handle to opened MVL file as generated by mvl_open()

Value

None

See Also

mvl_open, mvl_remap

mvl_compute_repeats *Find stretches of repeated rows among vectors*

Description

This function is passed a list of vector like MVL_OBJECTs which are considered as columns in a table. It returns a vector V starting with 1 and ending with number of rows plus 1, so that stretches of repeated rows can be found as V[i]:V[i+1]

Usage

```
mvl_compute_repeats(L)
```

Arguments

L	list of vector like MVL_OBJECTs

Value

partition describing repeated rows

mvl_extent_index_lapply *Apply function to indices of rows with matching hashes*

Description

Please use generic function `mvl_index_lapply()` instead.

Usage

```
mvl_extent_index_lapply(extent_index, data_list, fn)
```

Arguments

extent_index	MVL_OBJECT computed by `mvl_write_extent_index()`
data_list	a list of vectors of equal length. They can be MVL_OBJECTs or R vectors. If missing, scan the entire table one hash at a time.
fn	a function of two arguments - and index into `data_list` and a corresponding list of indices

Details

This function is passed the index computed by `mvl_write_extent_index()` and a list of vectors, which rows are used to compute 64-bit hashes. For each row, we call the function `fn(i, idx)`, where `i` gives the index of query row, and `idx` gives the indices of with matching hashes.

64-bit hashes have very few collisions, nevertheless the user is advised to double check that the values actually match.

The hash computation is type dependent, so 1 stored as an integer will produce a different hash than when stored as floating point. This function accounts for this by internally converting to types the index was generated with.

Value

a list of results of function `fn`

See Also

`mvl_index_lapply`, `mvl_group`

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=(1:100) %% 10), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_extent_index(Mtmp, list(Mtmp$df1[,"y",ref=TRUE]), "df1_extent_index_y")
Mtmp<-mvl_remap(Mtmp)
mvl_extent_index_lapply(Mtmp["df1_extent_index_y", ref=TRUE], list(c(2, 3)),
                                          function(i, idx) { return(list(i, idx))})
# Example of full scan
mvl_extent_index_lapply(Mtmp["df1_extent_index_y", ref=TRUE], ,
                                          function(i, idx) { return(list(i, idx))})

## End(Not run)
```

`mvl_find_matches` *Find matching rows*

Description

This function is passed two lists of MVL vectors which are interpreted in data.frame fashion. The indices of pairwise matches are returned in order of the arguments ("index1" and "index2"). In addition we return indices describing stretches with "index1" value constant (stretch_index1[i] to stretch_index1[i+1]-1)

Usage

```
mvl_find_matches(L1, L2, indices1 = NULL, indices2 = NULL)
```

Arguments

L1	list of vector like MVL_OBJECTs
L2	list of vector like MVL_OBJECTs
indices1	list of indices into objects to sort. If absent or NULL it is assumed to be from 1 to the length of the object.
indices2	list of indices into objects to sort. If absent or NULL it is assumed to be from 1 to the length of the object.

Value

A list of matches and match stretches

See Also

mvl_hash_vectors, mvl_order_vectors, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=rep(c("a", "b"), 50), y=1:100), "df1")
mvl_write_object(Mtmp, data.frame(x=rep(c("b", "c"), 50), y=21:120), "df2")
Mtmp<-mvl_remap(Mtmp)
L<-mvl_find_matches(list(Mtmp$df1[,"x",ref=TRUE], Mtmp$df1[,"y", ref=TRUE]),
                        list(Mtmp$df2[,"x",ref=TRUE], Mtmp$df2[,"y", ref=TRUE]))

## End(Not run)
```

---

mvl_fused_write_objects

Concatenate objects and write result into MVL file.

Description

This function can concatenate a mixture of R and MVL objects. For vectors it is the equivalent of c(). For array and matrices it works as cbind() For data frames it works as rbind, but row names are always dropped.

Usage

```
mvl_fused_write_objects(MVLHANDLE, L, name = NULL, drop.rownames = TRUE)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
L	a list of suitable R objects (vector, array, data.frame) or equivalent MVL objects.
name	if specified add a named entry to MVL file directory
drop.rownames	set to TRUE to prevent rownames from being written

Value

any object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, runif(100), "vec1")
mvl_write_object(Mtmp, runif(100), "vec2")
Mtmp<-mvl_remap(Mtmp)
mvl_fused_write_objects(Mtmp, list(Mtmp["vec1", ref=TRUE], Mtmp["vec2", ref=TRUE], runif(3)),
                                                                                name="vec3")

## End(Not run)
```

---

mvl_get_groups *Retrieve indices belonging to one or more groups*

Description

This function is passed the prev vector computed by mvl_write_groups and one or more indices from the first vector.

Usage

```
mvl_get_groups(prev, first_indices)
```

Arguments

prev	MVL_OBJECT prev computed by mvl_write_groups
first_indices	indices from first vector computed by mvl_write_groups

Value

a vector of indices

See Also

mvl_group

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=1:100), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_groups(Mtmp, list(Mtmp$df1[,"x",ref=TRUE], Mtmp$df1[,"y", ref=TRUE]), "df1_groups")
Mtmp<-mvl_remap(Mtmp)
print(mvl_get_groups(Mtmp["df1_groups", ref=TRUE]["prev", ref=TRUE], Mtmp$df1_groups$first[1:5]))

## End(Not run)
```

---

mvl_get_neighbors *Retrieve indices of nearby rows.*

Description

This function is passed the index computed by `mvl_write_spatial_index1` and a list of vectors, which rows are interpreted as points. For each row, the function returns a vector of indices describing rows that are close to it.

Usage

```
mvl_get_neighbors(spatial_index, data_list)
```

Arguments

`spatial_index`	MVL_OBJECT computed by `mvl_write_spatial_index1`
`data_list`	a list of vectors of equal length. They can be MVL_OBJECTs or R vectors.

Value

a list of vectors of indices

See Also

`mvl_write_spatial_index1`, `mvl_index_lapply`

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=1:100), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_spatial_index1(Mtmp, list(Mtmp$df1[,"x",ref=TRUE], Mtmp$df1[,"y", ref=TRUE]),
                                                              c(2, 3), "df1_sp_groups")
Mtmp<-mvl_remap(Mtmp)
print(mvl_get_neighbors(Mtmp["df1_sp_groups", ref=TRUE], list(c(0.5, 0.6), c(2, 3))))

## End(Not run)
```

mvl_group *Group identical rows*

Description

This function groups identical rows. The result is formatted as two vectors `stretch_index` and `index` Vector `index` contains stretches of indices with identical rows. Vector `stretch_index` describes stretches as stretch_index[i] to stretch_index[i+1]-1 This allows fast iteration over indices without creating excessive numbers of R objects when group sizes are small.

Usage

```
mvl_group(L, indices = NULL)
```

Arguments

`L`	list of vector like MVL_OBJECTs
`indices`	list of indices into objects to group. If absent or NULL it is assumed to be from 1 to the length of the object.

Value

A list of groups and group stretches

See Also

`mvl_group_lapply`, `mvl_hash_vectors`, `mvl_find_matches`, `mvl_order_vectors`, `mvl_find_matches`, `mvl_indexed_copy`, `mvl_merge`

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=rep(c("a", "b"), 50), y=(1:100)/5), "df1")
Mtmp<-mvl_remap(Mtmp)
df1<-Mtmp["df1", ref=TRUE]
G<-mvl_group(list(df1[,"x",ref=TRUE], df1[,"y", ref=TRUE]))
mvl_group_lapply(G, function(idx) { return(sum(df1[idx, "y"]))})

## End(Not run)
```

mvl_group_lapply *Apply function to index stretches*

Description

Iteratively call function `fn(idx)` over index stretches previously computed with `mvl_group`

Usage

```
mvl_group_lapply(G, fn)
```

Arguments

`G`	a list of groups and group stretches produced by `mvl_group`
`fn`	a function of one argument - list of indices

Value

a list of results of function `fn`

See Also

`mvl_group`

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=rep(c("a", "b"), 50), y=(1:100)/5), "df1")
Mtmp<-mvl_remap(Mtmp)
df1<-Mtmp$df1
G<-mvl_group(list(df1[,"x",ref=TRUE], df1[,"y", ref=TRUE]))
mvl_group_lapply(G, function(idx) { return(sum(df1[idx, "y"]))})

## End(Not run)
```

mvl_hash_vectors *Return hash values for each row*

Description

This function is passed a list of MVL vectors which are interpreted in data.frame fashion. For each row, i.e. set of vector values with the same index we compute a hash value. Identical rows produce identical hash values. The hash values have good entropy and can be used to map row values into random numbers.

Usage

```
mvl_hash_vectors(L, indices = NULL)
```

Arguments

L	list of vector like MVL_OBJECTs
indices	list of indices into objects to sort. If absent or NULL it is assumed to be from 1 to the length of the object.

Value

hash values in numeric format, with 52 valid bits. Each value is uniform between 1 and 2.

See Also

mvl_order_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge, mvl_write_hash_vectors

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, runif(100), "vec1")
Mtmp<-mvl_remap(Mtmp)
hash1<-mvl_hash_vectors(list(Mtmp["vec1", ref=TRUE]))

## End(Not run)
```

---

mvl_indexed_copy *Index copy vector*

Description

This function creates new MVL vectors and data frames by copying only rows or values specified by given indices. The vector indices can be an R integer or numeric vector, a logical vector of the size matching to the object being copied, or a suitable vector stored in MVL file.

Usage

```
mvl_indexed_copy(MVLHANDLE, x, indices, name = NULL, only.columns = NULL)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
x	a vector-like MVL_OBJECT or a data.frame stored in MVL file
indices	a vector of indices into x
name	if specified add a named entry to MVL file directory
only.columns	if x is MVL_OBJECT with class data.frame copy only columns specified in this character or integer vector

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_hash_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_order_vectors, mvl_merge, mvl_write_object, mvl_fused_write_objects

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, runif(100), "vec1")
Mtmp<-mvl_remap(Mtmp)
permutation1<-mvl_order_vectors(list(Mtmp["vec1", ref=TRUE]))
mvl_indexed_copy(Mtmp, Mtmp["vec1", ref=TRUE], permutation1, name="vec1_sorted")
Mtmp<-mvl_remap(Mtmp)
print(Mtmp$vec1_sorted)

## End(Not run)
```

---

mvl_index_lapply *Apply function to indices of nearby rows*

Description

This function is passed the index computed by `mvl_write_spatial_index1` or `mvl_write_extent_index` and a list of vectors, which are interpreted in a data frame fashion, or an R data.frame. For each row we retrieve that set of indices that matches it and call function fn(i, idx) with index i of row being processed and vector idx listing matched indices.

Usage

```
mvl_index_lapply(index, data_list, fn)
```

Arguments

`index`	MVL_OBJECT computed by `mvl_write_spatial_index1` or `mvl_write_extent_index`
`data_list`	a list of vectors of equal length. They can be MVL_OBJECTs or R vectors, or a data.fame.
`fn`	a function of two arguments - and index into `data_list` and a corresponding list of indices

Details

The notion of "matched indices" is specific to the type of index being used.

For an index created with `mvl_write_spatial_index1` we return the indices of nearby rows. The user should apply an additional cut to narrow down to actual indices needed.

For an index created with `mvl_write_extent_index` we return the indices of rows with identical hashes. Even though 64-bit hashes produce very few collisions, it is recommended to apply additional cut to ensure that only the exactly matching rows are returned.

Value

a list of results of function `fn`

See Also

`mvl_group`

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=1:100), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_spatial_index1(Mtmp, list(Mtmp$df1[,"x",ref=TRUE], Mtmp$df1[,"y", ref=TRUE]),
                                                                    c(2, 3), "df1_sp_groups")
Mtmp<-mvl_remap(Mtmp)
mvl_index_lapply(Mtmp["df1_sp_groups", ref=TRUE], list(c(0.5, 0.6), c(2, 3)),
                                          function(i, idx) { return(list(i, idx))})

## End(Not run)
```

`mvl_inherits` *Check inheritance of R or MVL objects*

Description

This function works just like the usual R `inherits()`, except that for MVL_OBJECTS it used the class value stored in the MVL file. For non-MVL objects the function simply calls the usual R `inherit()`, so it can be used instead of `inherit()` for code that operates on both usual R objects and MVL objects.

Usage

```
mvl_inherits(x, clstr, which = FALSE)
```

Arguments

x	any object
clstr	classes to match against
which	when TRUE return a boolean array indicating of which classes named in clstr are inherited by x. When FALSE return a single boolean indicating inheritance of any class named in clstr.

Value

character string giving object class

---

mvl_merge *Merge two MVL data frames and write the result*

Description

Merge two MVL data frames and write the result

Usage

```
mvl_merge(
  MVLHANDLE,
  df1,
  df2,
  name = NULL,
  by = NULL,
  by.x = by,
  by.y = by,
  suffixes = c(".x", ".y"),
  only.columns.x = NULL,
  only.columns.y = NULL
)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
df1	a data.frame stored in MVL file
df2	a data.frame stored in MVL file
name	if specified add a named entry to MVL file directory
by	list of columns to use as key
by.x	list of columns to use as key for df1
by.y	list of columns to use as key for df1
suffixes	rename columns with identical names using these suffixes
only.columns.x	only copy these columns from df1
only.columns.y	only copy these columns from df2

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_hash_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_order_vectors, mvl_fused_write_objects

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=rep(c("a", "b"), 50), y=1:100), "df1")
mvl_write_object(Mtmp, data.frame(x=rep(c("b", "c"), 50), y=runif(100), z=21:120), "df2")
Mtmp<-mvl_remap(Mtmp)
mvl_merge(Mtmp, Mtmp$df1, Mtmp$df2, by.x="y", by.y="z", only.columns.y=c("x"), name="df_merged")
Mtmp<-mvl_remap(Mtmp)
print(Mtmp$df_merged[1:10,])

## End(Not run)
```

---

mvl_neighbors_lapply *Apply function to indices of nearby rows*

Description

Please use generic function `mvl_index_lapply()` instead.

Usage

```
mvl_neighbors_lapply(spatial_index, data_list, fn)
```

Arguments

`spatial_index`	MVL_OBJECT computed by `mvl_write_spatial_index1`
`data_list`	a list of vectors of equal length. They can be MVL_OBJECTs or R vectors.
`fn`	a function of two arguments - and index into `data_list` and a corresponding list of indices

Details

This function is passed the index computed by `mvl_write_spatial_index1` and a list of vectors, which rows are interpreted as points. For each row, we call the function `fn(i, idx)`, where `i` gives the index of query row, and `idx` gives the indices of nearby rows.

Value

a list of results of function `fn`

See Also

`mvl_group`

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=1:100), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_spatial_index1(Mtmp, list(Mtmp$df1[,"x",ref=TRUE], Mtmp$df1[,"y", ref=TRUE]),
                                                                    c(2, 3), "df1_sp_groups")
Mtmp<-mvl_remap(Mtmp)
mvl_neighbors_lapply(Mtmp["df1_sp_groups", ref=TRUE], list(c(0.5, 0.6), c(2, 3)),
                                          function(i, idx) { return(list(i, idx))})

## End(Not run)
```

`mvl_object_stats` *Return MVL object properties*

Description

Provide detailed information on stored MVL object without retrieving it

Usage

```
mvl_object_stats(MVLHANDLE, offset = NULL, scan = FALSE)
```

Arguments

`MVLHANDLE`	either a handle provided by mvl_open() or an MVL object such as produced by indexing operators
`offset`	offset to the object which properties are to be retrieved
`scan`	scan vector element to obtain additional statistics

Details

This function is given either an MVL handle and an offset in MVL file to examine, or just a single parameter of class MVL_OBJECT that contains both handle and offset

This function returns a list of object parameters describing total number of elements, element type (as used by libMVL) and a pointer to the underlying data. The pointer is passed via a cast to double to preserve its 64-bit value and can be used with custom C code, for example by using package inline.

Value

list with object properties

`mvl_open` *Open an MVL file*

Description

Open an MVL format file for reading and/or writing.

Usage

```
mvl_open(filename, append = FALSE, create = FALSE)
```

Arguments

`filename`	path to file.
`append`	specify TRUE when you intend to write data into the file
`create`	when TRUE create file if it did not exist

Details

MVL stands for "Mapped vector library" and is a file format designed for efficient memory mapped access. An MVL file can be much larger than physical memory of the machine.

`mvl_open` returns a handle that can be used to access MVL files. Files opened read-only are memory mapped and do not use a file descriptor, and thus are not subject to limits on the number of open files. Files opened for writing data do use a file descriptor. Once opened for read access the data can be accessed using usual R semantics for lists, data.frames and arrays.

Value

handle to opened MVL file

See Also

`mvl_close`, `mvl_remap`

Examples

```
## Not run:
M1<-mvl_open("test1.mvl", append=TRUE, create=TRUE)
mvl_write_object(M1, data.frame(x=1:2, y=rnorm(2)), "test_frame")
mvl_close(M1)

M2<-mvl_open("test1.mvl")
print(names(M2))
print(M2["test_frame"])
```

```
mvl_close(M2)

M3<-mvl_open("test2.mvl", append=TRUE, create=TRUE)
L<-list()
df<-data.frame(x=1:1e6, y=rnorm(1e6), s=rep(c("a", "b"), 5e5))
L["x"]]<-mvl_write_object(M3, df, drop.rownames=TRUE)
L["description"]]<-"Example of large data frame"
mvl_write_object(M3, L, "test_object")
mvl_close(M3)

M4<-mvl_open("test2.mvl")
print(names(M4))
L<-M4["test_object"]
print(L)
print(L[["x"]][1:20,])
mvl_object_stats(L[["x"]])
# If you need to get the whole x, one can use mvl2R(L[["x"]])
mvl_close(M4)

## End(Not run)
```

`mvl_order_vectors` *Return permutation sorting vector entries*

Description

This function is similar to R order() function, but operates on MVL_OBJECTS.

Usage

```
mvl_order_vectors(
  L,
  indices = NULL,
  decreasing = FALSE,
  sort_function = ifelse(decreasing, 2, 1)
)
```

Arguments

`L`	list of vector like MVL_OBJECTs
`indices`	list of indices into objects to sort. If absent or NULL it is assumed to be from 1 to length of the object.
`decreasing`	whether to sort in ascending or decreasing order. This parameter is provided for compatibility with `order()` function
`sort_function`	specifies desired sort order

Value

sorted indices

See Also

mvl_hash_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, runif(100), "vec1")
Mtmp<-mvl_remap(Mtmp)
permutation1<-mvl_order_vectors(list(Mtmp["vec1", ref=TRUE]))

## End(Not run)
```

---

mvl_remap *Enlarge memory map to include recently loaded data.*

Description

This function operates on MVL files opened for writing. When writing new data to the MVL file that data is appended at the end and past the end of previously mapped data. Calling `mvl_remap()` updates the memory mapping to include all the data written before mvl_remap() was called. The MVL file directory is also updated to include recently added entries. Old handles can still be used, but will not include updated directory information. MVL_OBJECT's previously obtained from this handle continue to be valid.

Usage

```
mvl_remap(MVLHANDLE, append = TRUE)
```

Arguments

MVLHANDLE	handle to opened MVL file as generated by `mvl_open()` or `mvl_remap()`
append	specify FALSE when you do not intend to write the file.

Details

`mvl_remap` returns a handle with updated directory.

Value

handle to MVL file, with updated directory.

See Also

mvl_open, mvl_close

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, runif(100), "vec1")
Mtmp<-mvl_remap(Mtmp)
print(Mtmp["vec1"])

## End(Not run)
```

`mvl_start_write_vector` *Piecewise output of very long numeric and integer vectors*

Description

While `mvl_fused_write_objects` can be used to create very large vectors and data frames of arbitrary type, it requires piecewise data to be written first into an MVL file. Functions `mvl_start_write_vector()` and `mvl_rewrite_vector()` provide a way to create very long vectors in one pass. Only numeric and integer vectors are supported.

Usage

```
mvl_start_write_vector(MVLHANDLE, x, expected.length = NULL, name = NULL)

mvl_rewrite_vector(obj, offset, x)
```

Arguments

`MVLHANDLE`	handle to opened MVL file as generated by mvl_open()
`x`	an integer or numeric vector
`expected.length`	the length of vector to create. Use double to pass large values
`name`	if specified add a named entry to MVL file directory
`obj`	an MVL vector object to modify
`offset`	the offset into MVL vector (starting with 1) to write x

Details

One convenient use is to compute `f(x,y,z,...)` with very long vector arguments by iterating over indices. The iteration can be done using fixed blocks of indices, or by using groups of indices computed with other MVL functions.

It is generally recommended to call `mvl_rewrite_vector()` with large blocks to improve I/O performance and reduce number of writes to underlying media.

See Also

mvl_fused_write_objects

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
offset<-mvl_start_write_object(Mtmp, runif(10), expected.length=1000, "vec1")
Mtmp<-mvl_remap(Mtmp)
mvl_rewrite_vector(Mtmp[offset], 50, rnorm(20))

## End(Not run)
```

---

mvl_status *Return status of MVL package*

Description

Return status of MVL package

Usage

```
mvl_status()
```

Value

list of status values

---

mvl_write_extent_index *Compute and write extent index*

Description

This function computes a hash-based index that allows to find indices of rows which hashes match query values. While it can be applied to arbitrary data, it is optimized for the common case when vectors contain stretches of repeated values describing row groups to be processed. This is particularly relevant for R because vectorized processing of row batches is the only practical way to scan very large tables using pure-R code.

Usage

```
mvl_write_extent_index(MVLHANDLE, L, name = NULL)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
L	list of vector like MVL_OBJECTs
name	if specified add a named entry to MVL file directory

Details

mvl_write_extent_index() creates the index in memory and then writes it out. The memory usage is proportional to the number of repeat stretches. Sorting tables improves performance, but is not a requirement.

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_order_vectors, mvl_index_lapply, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge, mvl_hash_vectors, mvl_get_groups

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=(1:100) %% 10), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_extent_index(Mtmp, list(Mtmp$df1[,"y",ref=TRUE]), "df1_extent_index_y")
Mtmp<-mvl_remap(Mtmp)
mvl_index_lapply(Mtmp["df1_extent_index_y", ref=TRUE], list(c(2, 3)),
                                          function(i, idx) { return(list(i, idx))})
# Example of full scan
mvl_index_lapply(Mtmp["df1_extent_index_y", ref=TRUE], ,
                                          function(i, idx) { return(list(i, idx))})

## End(Not run)
```

---

mvl_write_groups *Write group information for each row*

Description

This function is passed a list of MVL vectors which are interpreted in data.frame fashion. These rows are split into groups so that identical rows are guaranteed to belong to the same group. This is done internally based on 20-bit hash values. This function is convenient to use as a way to partition very large datasets before applying mvl_group or mvl_find_matches. The groups can be obtained by using mvl_get_groups

Usage

```
mvl_write_groups(MVLHANDLE, L, name = NULL)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
L	list of vector like MVL_OBJECTs
name	if specified add a named entry to MVL file directory

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_order_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge, mvl_hash_vectors, mvl_get_groups

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=1:100), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_groups(Mtmp, list(Mtmp$df1[,"x",ref=TRUE], Mtmp$df1[,"y", ref=TRUE]), "df1_groups")
Mtmp<-mvl_remap(Mtmp)
print(mvl_get_groups(Mtmp["df1_groups", ref=TRUE]["prev", ref=TRUE], Mtmp$df1_groups$first[1:5]))

## End(Not run)
```

---

mvl_write_hash_vectors

Write hash values for each row

Description

This function is passed a list of MVL vectors which are interpreted in data.frame fashion. For each row, i.e. set of vector values with the same index we compute a 64-bit hash value. Identical rows produce identical hash values. The hash values are written into 64-bit integer vector. This function is meant for use with data that is too large to handle comfortably.

Usage

```
mvl_write_hash_vectors(MVLHANDLE, L, name = NULL)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
L	list of vector like MVL_OBJECTs
name	if specified add a named entry to MVL file directory

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_order_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge, mvl_hash_vectors

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, runif(100), "vec1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_hash_vectors(Mtmp, list(Mtmp["vec1", ref=TRUE]), "vec1_hash")
Mtmp<-mvl_remap(Mtmp)
print(length(Mtmp["vec1_hash"]))

## End(Not run)
```

---

mvl_write_object *Write R object into MVL file*

Description

Write R object into MVL file

Usage

```
mvl_write_object(MVLHANDLE, x, name = NULL, drop.rownames = FALSE)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
x	a suitable R object (vector, array, list, data.frame) or a vector-like MVL_OBJECT
name	if specified add a named entry to MVL file directory
drop.rownames	set to TRUE to prevent rownames from being written

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_indexed_copy, mvl_merge

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, runif(100), "vec1")
L<-list()
L["x"]]<-mvl_write_object(Mtmp, 1:5)
L["y"]]<-mvl_write_object(Mtmp, c("a", "b"))
L["df"]]<-mvl_write_object(Mtmp, data.frame(x=1:100, z=runif(100)))
mvl_write_object(Mtmp, L, "L")
Mtmp<-mvl_remap(Mtmp)
print(Mtmp$L)

## End(Not run)
```

---

mvl_write_serialized_object

Write R object in serialized form

Description

This function packages the object into a raw vector before writing it out. The raw vector is tagged with special class that assures the object is automatically converted back to R representation when reading. Serialized objects can only be read completely.

Usage

```
mvl_write_serialized_object(MVLHANDLE, x, name = NULL)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
x	a suitable R object (vector, array, list, data.frame) or a vector-like MVL_OBJECT
name	if specified add a named entry to MVL file directory

Details

This function can be used in rare cases when it is important to store a complete R object, but it is not important for it to be accessible by other programs, and it is not important to conserve memory or bandwidth.

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_write_object

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_serialized_object(Mtmp, lm(rnorm(100)~runif(100)), "LM1")
Mtmp<-mvl_remap(Mtmp)
print(mvl2R(Mtmp$LM1))

## End(Not run)
```

---

mvl_write_spatial_groups

Write spatial group information for each row

Description

Please use mvl_write_spatial_index1() instead.

Usage

```
mvl_write_spatial_groups(MVLHANDLE, L, bits, name = NULL)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
L	list of vector like MVL_OBJECTs
bits	a vector of bit values to use for each member of L
name	if specified add a named entry to MVL file directory

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_order_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge, mvl_hash_vectors, mvl_get_groups

mvl_write_spatial_index1 *Write spatial group information for each row*

Description

This function is passed a list of MVL vectors which are interpreted in data.frame fashion. These rows are split into groups so that identical rows are guaranteed to belong to the same group. This is done using partition into equal sized bins. This function is meant for constructing spatial indexes.

Usage

```
mvl_write_spatial_index1(MVLHANDLE, L, bits, name = NULL)
```

Arguments

MVLHANDLE	a handle to MVL file produced by mvl_open()
L	list of vector like MVL_OBJECTs
bits	a vector of bit values to use for each member of L
name	if specified add a named entry to MVL file directory

Value

an object of class MVL_OFFSET that describes an offset into this MVL file. MVL offsets are vectors and can be concatenated. They can be written to MVL file directly, or as part of another object such as list.

See Also

mvl_order_vectors, mvl_find_matches, mvl_group, mvl_find_matches, mvl_indexed_copy, mvl_merge, mvl_hash_vectors, mvl_get_groups

Examples

```
## Not run:
Mtmp<-mvl_open("tmp_a.mvl", append=TRUE, create=TRUE)
mvl_write_object(Mtmp, data.frame(x=runif(100), y=1:100), "df1")
Mtmp<-mvl_remap(Mtmp)
mvl_write_spatial_index1(Mtmp, list(Mtmp$df1[,"x",ref=TRUE], Mtmp$df1[,"y", ref=TRUE]),
                                                              c(2, 3), "df1_sp_groups")
Mtmp<-mvl_remap(Mtmp)
print(mvl_get_neighbors(Mtmp["df1_sp_groups", ref=TRUE], list(c(0.5, 0.6), c(2, 3))))

## End(Not run)
```

mvl_xlength *Return length of MVL or R vector as a numeric value*

Description

Internally this calls R function xlength() rather than length(). This allows to obtain length of larger vectors. For MVL vectors this returns the length of the vector.

Usage

```
mvl_xlength(x)
```

Arguments

x	any R object

Value

length of object as as numeric value

names.MVL *Print MVL directory*

Description

Print MVL directory

Usage

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

Arguments

x	handle to MVL file as created by mvl_open

Value

character vector of names present in the directory

names.MVL_OBJECT *Retrieve MVL object names*

Description

Retrieve MVL object names

Usage

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

Arguments

x	MVL_OBJECT as retrieved by subscription operators

Value

character vector of names

print.MVL *Print MVL*

Description

Print MVL

Usage

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

Arguments

x	handle to MVL file as created by mvl_open
...	not used.

Value

invisible(MVLHANDLE)

print.MVL_OBJECT *Print MVL object This is a convenience function for displaying MVL_OBJECTs.*

Description

Print MVL object This is a convenience function for displaying MVL_OBJECTs.

Usage

```
## S3 method for class 'MVL_OBJECT'
print(x, ..., small_length = 10)
```

Arguments

x	MVL_OBJECT as retrieved by subscription operators
small_length	do not list more than this number of columns in data frames
...	not used.

Value

invisible(obj)

[.MVL *MVL handle subscription operator*

Description

Retrieve objects stored in mappable vector library

Usage

```
## S3 method for class 'MVL'
MVLHANDLE[y, raw = FALSE, ref = FALSE, drop = TRUE]
```

Arguments

MVLHANDLE	- handle to opened MVL file as generated by mvl_open
y	- name of object to retrieve
raw	- request to return data in raw format when it does not map exactly to R data types.
ref	- always return an MVL_OBJECT
drop	- whether to drop dimensionality, such as when a sublist contains only one element

Details

See mvl_open for example.

Value

Stored object

[.MVL_OBJECT *MVL object subscription operator*

Description

Retrieve objects stored in mappable vector library. Large nested objects are returned as instances of MVL_OBJECT to delay access until needed.

Usage

```
## S3 method for class 'MVL_OBJECT'
obj[i, ..., drop = TRUE, raw = FALSE, recurse = FALSE, ref = FALSE]
```

Arguments

obj	- MVL object retrieved by subscription of MVL library or other objects
i	- optional index.
drop	- whether to drop dimensionality, such as done with R array or data frames
raw	- request to return data in raw format when it does not map exactly to R data types.
recurse	- force recursive conversion to pure R objects.
ref	- always return an MVL_OBJECT
...	optional additional indices for multidimensional arrays and data frames

Details

See mvl_open for example.

Value

Stored object

[[.MVL_OBJECT *MVL object subscription operator*

Description

Retrieve objects stored in mappable vector library. Large nested objects are returned as instances of MVL_OBJECT to delay access until needed.

Usage

```
## S3 method for class 'MVL_OBJECT'
obj[[i, raw = FALSE, recurse = FALSE, ref = FALSE]]
```

Arguments

obj	- MVL object retrieved by subscription of MVL library or other objects
i	- index.
raw	- request to return data in raw format when it does not map exactly to R data types.
recurse	- force recursive conversion to pure R objects.
ref	- always return an MVL_OBJECT

Details

See mvl_open for example.

Value

Stored object

$.MVL *MVL handle subscription operator*

Description

Retrieve objects stored in the library. Unlike for R lists the match on name is always exact.

Usage

```
## S3 method for class 'MVL'
MVLHANDLE$name
```

Arguments

MVLHANDLE	- handle to opened MVL file as generated by mvl_open
name	- name of object to retrieve

Value

Stored object

Index