Package: Rfmalloc (via r-universe)

July 21, 2026

Title Out-of-Core Arrays with Pluggable Codecs and Compute Backends

Version 0.1.0

Description Computes on arrays that do not fit in memory. Vectors, matrices and tensors are allocated inside a memory-mapped file through R's ALTREP interface, backed by a patched copy of the 'fmalloc' allocator, so available RAM becomes a speed gradient rather than a hard limit. Two plugin registries sit on that substrate: a codec registry deciding how bytes are stored, from lossless floating-point and sparse encodings to compressed and quantized blocks, and a matrix-multiply backend registry deciding how products are computed, defaulting to whatever 'BLAS' the R build links against. Matrix products stream in bounded panels and release the pages they have consumed, so operands larger than memory still multiply. A registered backend may decline any product, in which case the default path answers it, so results never depend on which backend is selected. Also provides persistent and scratch runtimes, durable reference-based serialization, explicit vector lifecycle management, multiple runtime handles per session, and out-of-core reductions and principal components for genomics-scale matrices.

License GPL (>= 2)

Encoding UTF-8

Imports methods

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests bench, kalis, lobstr, tinytest

SystemRequirements GNU make, POSIX threads

URL https://github.com/sounkou-bioinfo/Rfmalloc, https://sounkou-bioinfo.github.io/Rfmalloc/Rfmalloc/

Depends R (>= 4.4.0)

BugReports https://github.com/sounkou-bioinfo/Rfmalloc/issues

Config/pak/sysreqs make

Repository https://rgenomicsetl.r-universe.dev

Date/Publication 2026-07-18 21:34:09 UTC

RemoteUrl https://github.com/RGenomicsETL/Rfmalloc

RemoteRef HEAD

RemoteSha 6e05ad03613e89cab2f8bf8eb3a49d553d5a3bed

RemoteSubdir packages/Rfmalloc

Contents

`Rfmalloc-package` *Rfmalloc: Memory-Mapped File Allocation for R*

Description

Rfmalloc provides experimental memory-mapped file allocation capabilities for R using a patched copy of the fmalloc library. The current package exposes ALTREP file-backed vector allocation for logical, integer, numeric, raw, complex, character, and list vectors with fmalloc payload storage.

Main Functions

`open_fmalloc` Open an explicit fmalloc runtime handle.

`init_fmalloc` Open and install a default fmalloc runtime.

`create_fmalloc_vector` Create vectors using fmalloc.

`create_fmalloc_matrix` Create matrix-shaped fmalloc vectors.

`create_fmalloc_array` Create array-shaped fmalloc vectors.

`create_fmalloc_data_frame` Create data.frames from fmalloc-backed columns.

`as_fmalloc_matrix` Convert fmalloc vectors to matrix-shaped objects.

`as_fmalloc_array` Convert fmalloc vectors to array-shaped objects.

`as_fmalloc_data_frame` Convert fmalloc-backed objects to a data.frame.

`fmalloc_linalg` Matrix products for fmalloc-backed vectors and matrices.

`fmalloc_tensor` Typed and ALP-compressed tensors with panel-streamed products.

`fmalloc_matmul_ooc` Out-of-core matrix products for matrices larger than RAM.

`fmalloc_insitu` In-place, by-reference mutation that bypasses copy-on-modify.

`fmalloc_backend` Select a pluggable matrix-multiply backend (default R's BLAS).

`fmalloc_pca` Out-of-core PCA and variance reductions for genomics (single-cell) workflows.

`list_fmalloc_allocations` List persistent allocation catalog records.

`diagnose_fmalloc_runtime` Summarize persistent allocation catalog state and runtime diagnostics.

`fmalloc_api` Inspect runtimes and vectors through the public R/native API.

`cleanup_fmalloc` Request cleanup of an fmalloc runtime.

Current Scope

- ALTREP file-backed allocation for logical, integer, numeric, raw, complex, character, and list vectors. List elements are restricted to `NULL` or Rfmalloc-backed vectors from the same runtime.
- Large allocations spanning multiple fmalloc chunks.
- Multiple runtime handles in one R process.

- Persistent and scratch runtime modes.
- Reference serialization for persistent fixed-width atomic and character ALTREP vectors.
- Fmalloc-backed ALTREP subset copies for vector indexing operations.
- Elementwise arithmetic/comparison/logical `Ops` and fmalloc-backed `matrixOps` products for `%*%`, `crossprod()`, and `tcrossprod()`.
- An in-file allocation catalog for persistent vectors.
- A C-callable API and installed header for other packages.
- Native lifetime tracking so runtime mappings outlive reachable vectors allocated from them.
- Runtime and catalog diagnostics for planning recovery and operational cleanup.
- R-facing API helpers for runtime/vector predicates, metadata, payload pointers, and version checks.

Known Limitations

- ALTREP-backed dispatch now covers core `Ops`, `Summary`, `Math`, `Math2`, matrix `rowSums`/`colSums`/`rowMeans`/`colMeans`, and initial `matrixOps` workflows through S3 methods for common vector/matrix usage.
- Explicit base-fallback boundaries are:
  - `rowSums()`, `colSums()`, `rowMeans()`, and `colMeans()` when the input is not an exact 2D matrix or `dims != 1L`; these cases now emit a warning and call the corresponding `base::` reducer.
  - Scalar or zero-length results from `Summary`, `Math`, and `Math2` generics (for example `sum(x)` returning a single value) are returned as ordinary R scalars by design.
  - Matrix products use BLAS `dgemm` for finite numeric operands; operands containing `NA`/`NaN`/`Inf` and logical/integer/complex operands are computed by managed native loops with base-consistent semantics.
- Full operator- and method-family coverage is still incomplete for all R generics. Some advanced families may still materialize ordinary R objects in a few edge cases.

Future Work

Future work includes complex `zgemm` and further LAPACK-backed kernels, view-based subset representations, catalog compaction and reset tooling, metadata storage for attributes on persisted elements, robust nested-list reference validation, and compaction of recovery metadata.

Author(s)

Maintainer: Sounkou Mahamane Toure <sounkoutoure@gmail.com>

Other contributors:

- Kenichi Yasukata (fmalloc) [copyright holder]
- Wolfram Gloger (ptmalloc3) [copyright holder]
- Free Software Foundation, Inc. (selected GNU C Library support files) [copyright holder]

See Also

Useful links:

- https://github.com/sounkou-bioinfo/Rfmalloc
- https://sounkou-bioinfo.github.io/Rfmalloc/Rfmalloc/
- Report bugs at https://github.com/sounkou-bioinfo/Rfmalloc/issues

---

`as_fmalloc_array` *Convert a vector to fmalloc array metadata*

Description

Returns an existing vector re-typed as an array by installing array dimensions (and optional dimnames) as metadata.

Usage

```
as_fmalloc_array(x, dim = NULL, dimnames = NULL, copy = TRUE)
```

Arguments

`x`	A vector.
`dim`	Target dimension vector.
`dimnames`	Optional `dimnames` for the resulting array.
`copy`	If TRUE (default), allocate a new fmalloc-backed array object. If FALSE, install metadata in place on the same fmalloc ALTREP payload without allocation (this also updates any aliases of `x`).

Value

An array object, backed by the same payload when `copy = FALSE`.

`as_fmalloc_data_frame` *Convert to data.frame for fmalloc vectors*

Description

Thin convenience wrapper around `data.frame()`.

Usage

```
as_fmalloc_data_frame(
  ...,
  row.names = NULL,
  check.names = TRUE,
  stringsAsFactors = FALSE
)
```

Arguments

`...`	Columns or objects to include in the frame.
`row.names`	Optional row names for the frame.
`check.names`	Whether to enforce syntactic column names.
`stringsAsFactors`	Deprecated: retained for compatibility.

Value

A `data.frame` containing the supplied columns.

`as_fmalloc_matrix` *Convert a vector to fmalloc matrix metadata*

Description

Returns an existing vector re-typed as a matrix by installing matrix dimensions (and optional dimnames) as metadata.

Usage

```
as_fmalloc_matrix(x, nrow = NULL, ncol = NULL, dimnames = NULL, copy = TRUE)
```

Arguments

`x`	A vector.
`nrow`	Optional target row count.
`ncol`	Optional target column count.
`dimnames`	Optional `dimnames` for the resulting matrix.
`copy`	If TRUE (default), allocate a new fmalloc-backed matrix object. If FALSE, install metadata in place on the same fmalloc ALTREP payload without allocation (this also updates any aliases of `x`).

Value

A matrix object, backed by the same payload when `copy = FALSE`.

`cleanup_fmalloc` *Clean Up fmalloc*

Description

Requests cleanup of an fmalloc runtime. If vectors allocated from the runtime are still reachable, the native mapping is kept alive until those vectors are garbage-collected.

Usage

```
cleanup_fmalloc(runtime = NULL)
```

Arguments

`runtime`	Optional runtime handle returned by `open_fmalloc()`. If not supplied, the current default runtime is cleaned up.

Value

NULL (invisibly)

Examples

```
## Not run:
init_fmalloc("data.bin")
v <- create_fmalloc_vector("integer", 100)
rm(v)
gc()
cleanup_fmalloc()

## End(Not run)
```

`create_fmalloc_array` *Create Array Using fmalloc*

Description

Creates an fmalloc-backed ALTREP array in a single step by allocating vector storage and installing array dimensions (and optional dimnames).

Usage

```
create_fmalloc_array(
  type = "integer",
  dim,
  dimnames = NULL,
  runtime = NULL,
  zero_initialize = TRUE
)
```

Arguments

type	Character string specifying the vector type. Supported values are the same as for create_fmalloc_vector().
dim	Integer dimension vector.
dimnames	Optional dimnames for the array.
runtime	Optional runtime handle returned by open_fmalloc(). If not supplied, the default runtime established by init_fmalloc() is used.
zero_initialize	Logical scalar passed through to payload allocation. See create_fmalloc_vector() for semantics.

Value

An fmalloc-backed ALTREP array.

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"))
a <- create_fmalloc_array("numeric", dim = c(2L, 3L), runtime = rt)
cleanup_fmalloc(rt)

## End(Not run)
```

---

create_fmalloc_data_frame

Construct data.frame from fmalloc columns

Description

Thin constructor wrapper around data.frame() that keeps fmalloc vectors as column payloads.

Usage

```
create_fmalloc_data_frame(
  ...,
  row.names = NULL,
  check.names = TRUE,
  stringsAsFactors = FALSE
)
```

Arguments

`...`	Columns to include in the frame.
`row.names`	Optional row names for the frame.
`check.names`	Whether to enforce syntactic column names.
`stringsAsFactors`	Deprecated: retained for compatibility.

Value

A `data.frame` with the provided columns.

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"))
x <- create_fmalloc_vector("integer", 3, runtime = rt)
y <- create_fmalloc_vector("character", 3, runtime = rt)
x[] <- 1:3
y[] <- c("a", "b", "c")
df <- create_fmalloc_data_frame(x = x, y = y)
cleanup_fmalloc(rt)

## End(Not run)
```

`create_fmalloc_matrix` *Create Matrix Using fmalloc*

Description

Creates an fmalloc-backed ALTREP matrix in a single step by allocating vector storage and installing matrix dimensions (and optional dimnames).

Usage

```
create_fmalloc_matrix(
  type = "integer",
  nrow,
  ncol,
  dimnames = NULL,
  runtime = NULL,
  zero_initialize = TRUE
)
```

Arguments

type	Character string specifying the vector type. Supported values are the same as for create_fmalloc_vector().
nrow	Integer number of rows.
ncol	Integer number of columns.
dimnames	Optional dimnames list for the matrix.
runtime	Optional runtime handle returned by open_fmalloc(). If not supplied, the default runtime established by init_fmalloc() is used.
zero_initialize	Logical scalar passed through to payload allocation. See create_fmalloc_vector() for semantics.

Value

An fmalloc-backed ALTREP matrix object.

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"))
m <- create_fmalloc_matrix("integer", nrow = 2, ncol = 3, runtime = rt)
cleanup_fmalloc(rt)

## End(Not run)
```

---

create_fmalloc_vector *Create Vector Using fmalloc*

Description

Creates an ALTREP vector using a file-backed fmalloc runtime. The returned object is ALTREP from creation time. Fixed-width atomic payload bytes are allocated directly with fmalloc, and ALTREP duplication and vector subsetting keep copy-on-write copies fmalloc-backed without using R's non-API Rf_allocVector3() path.

Usage

```
create_fmalloc_vector(
  type = "integer",
  length,
  runtime = NULL,
  zero_initialize = TRUE
)
```

Arguments

type	Character string specifying the vector type. Supported values are "logical", "integer", "numeric"/"double", "raw", "complex", "character", and "list". Fixed-width atomic types expose a direct writable fmalloc DATAPTR; character vectors store string bytes in fmalloc and materialize R CHARSXP values on demand; list vectors use ALTREP element access with an R-visible reference sidecar for GC safety and only accept NULL or fmalloc-backed vectors from the same runtime as elements. Persistent list containers are serialized by nested reference states when all elements are recoverable from the same runtime.
length	Non-negative integer-valued length of the vector to create. Supports exact values up to 2^52 in the R interface.
runtime	Optional runtime handle returned by open_fmalloc(). If not supplied, the default runtime established by init_fmalloc() is used.
zero_initialize	Logical scalar. If TRUE (default), newly allocated payload bytes are zero-initialized. Set FALSE to skip initialization for faster large allocations when you will fully initialize values yourself.

Value

A vector of the specified type and length, allocated using fmalloc.

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"))
v <- create_fmalloc_vector("integer", 1000, runtime = rt)
cleanup_fmalloc(rt)

## End(Not run)
```

---

destroy_fmalloc_vector

Explicitly destroy a fmalloc vector

Description

Releases runtime bookkeeping for a single fmalloc ALTREP vector immediately. In scratch mode, payload memory is immediately reclaimed. In persistent mode, the vector payload is retained by default so existing on-disk state remains durable; optional `unsafe = TRUE` reclaims payload memory and marks metadata as non-recoverable.

Usage

```
destroy_fmalloc_vector(x, unsafe = FALSE)
```

Arguments

`x`	Fmalloc ALTREP vector to destroy.
`unsafe`	Whether to physically free persistent payload bytes. Unsafe destroy is intended for short-lived scratch-like cleanup and will mark the catalog entry as non-recoverable.

Details

Explicit destroy fails when a vector is still referenced by another fmalloc list vector as a child.

Value

Logical value indicating whether a live vector was destroyed.

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"), mode = "persistent")
v <- create_fmalloc_vector("integer", 10, runtime = rt)
destroy_fmalloc_vector(v)

## End(Not run)
```

---

diagnose_fmalloc_runtime

Diagnose fmalloc runtime state

Description

Returns diagnostic metadata for an open runtime handle, including lightweight runtime attributes, the current allocation catalog, and a catalog-level summary useful for estimating reclaimable/fragmented payload regions.

Usage

```
diagnose_fmalloc_runtime(runtime = NULL)
```

Arguments

runtime	Optional runtime handle returned by open_fmalloc(). If not supplied, the current default runtime is used.

Value

A named list with three components:

- runtime: runtime metadata such as file path, UUID, mode, catalog counters, live vectors, and reference state;
- catalog: the full allocation catalog returned by list_fmalloc_allocations();
- summary: a compact set of computed diagnostics and an explicit compaction status note.

See Also

list_fmalloc_allocations()

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"), mode = "persistent")
x <- create_fmalloc_vector("integer", 4, runtime = rt)
y <- create_fmalloc_vector("logical", 2, runtime = rt)
diagnose_fmalloc_runtime(rt)
cleanup_fmalloc(rt)

## End(Not run)
```

---

fmalloc_api *Public Rfmalloc API helpers*

Description

Introspection and lifecycle helpers for fmalloc runtime handles and fmalloc ALTREP vectors. These functions are thin R wrappers around the package's native registered routines and mirror the installed C-callable API.

Usage

```
fmalloc_default_runtime()

is_fmalloc_runtime(x)

is_fmalloc_vector(x)

fmalloc_runtime(x)
```

```
fmalloc_runtime_info(runtime = NULL)

fmalloc_vector_info(x)

fmalloc_vector_type(x, label = TRUE)

fmalloc_vector_length(x)

fmalloc_vector_payload_ptr(x)
```

Arguments

x	An object to test or inspect. For vector helpers, x must be an active fmalloc ALTREP vector unless the function name starts with is_.
runtime	Optional runtime handle returned by open_fmalloc(). If not supplied, the current default runtime from init_fmalloc() is used.
label	Logical scalar. If TRUE, return an R type label such as "integer"; if FALSE, return the underlying R SEXPTYPE integer code.

Value

Depends on the helper: logical predicates, runtime external pointers, metadata lists, or payload external pointers.

---

fmalloc_backend *Pluggable matrix-multiply backend*

Description

Rfmalloc's matrix-product kernels (%*%, crossprod(), tcrossprod(), the out-of-core and typed-tensor products) dispatch their dgemm calls through a selectable backend. By default this is R's BLAS; downstream packages can register an alternative (for example a GPU cuBLAS kernel or an out-of-core-aware GEMM) through the Rfmalloc_register_matmul_backend C-callable and select it here. Selection is Rfmalloc-scoped: base R's %*% is unaffected.

Usage

```
fmalloc_matmul_backend(name = NULL)

fmalloc_matmul_backends()
```

Arguments

name	Backend name to select. "blas" (or NULL/"") selects the default BLAS path.

Details

A registered backend may decline a given call (returning non-zero), in which case Rfmalloc falls back to R's BLAS for that product.

Value

`fmalloc_matmul_backend()` returns the active backend name; `fmalloc_matmul_backends()` returns the registered backend names (BLAS is always available and not listed).

Examples

```
fmalloc_matmul_backend()      # "blas" by default
fmalloc_matmul_backends()     # names registered by other packages
```

`fmalloc_bed` *PLINK 1 genotypes as a 2-bit fmalloc tensor*

Description

Encodes an integer dosage matrix (`0`, `1`, `2`, `NA`) into PLINK 1 `.bed` bit packing and stores it in fmalloc-backed, memory-mapped storage as an fmalloc_tensor of codec `"bed"`. Samples are rows, variants are columns, matching `.bed`'s SNP-major layout: a variant is a contiguous column.

Usage

```
fmalloc_bed(x, runtime = NULL)
```

Arguments

`x`	An integer matrix of dosages of the first allele: `0`, `1`, `2`, or `NA`.
`runtime`	Runtime handle from `open_fmalloc()`; defaults to the runtime established by `init_fmalloc()`.

Details

Two bits per genotype. That is four times tighter than a one-byte-per-genotype file-backed matrix, and thirty-two times tighter than the doubles it decodes to. Products against the tensor decode bounded column panels on the fly and contract them with BLAS, so the genotypes are never materialized as doubles.

Missing genotypes decode to `NA_real_`, which the matrix-product path does not impute; standardize or impute before multiplying.

Value

An `fmalloc_tensor` of dtype `"bed"` with `dim(x)`.

See Also

create_fmalloc_tensor(), fmalloc_tensor_materialize()

Examples

```
rt <- open_fmalloc(tempfile(), size_gb = 0.1)
g <- matrix(c(0L, 1L, 2L, NA_integer_, 2L, 0L), nrow = 3, ncol = 2)
tn <- fmalloc_bed(g, runtime = rt)
fmalloc_tensor_materialize(tn)
cleanup_fmalloc(rt)
```

---

fmalloc_bed_standardize

Bake per-variant standardization into a bed tensor

Description

Computes each variant's mean and standard deviation in one streaming pass and stores them in the tensor, so that every subsequent decode returns standardized, mean-imputed values: missing genotypes decode to the variant mean, hence to `0` after centering. Products against the standardized tensor are therefore centered-and-scaled with no genotype ever materialized as a double and no second pass, which is what a genotype PCA or GRM needs.

Usage

```
fmalloc_bed_standardize(x, scale = c("sd", "binomial"), runtime = NULL)
```

Arguments

x	A "bed" fmalloc_tensor from fmalloc_bed() (raw, not already standardized).
scale	One of "sd" (default; the sample standard deviation of the mean-imputed column, matching scale()) or "binomial" (sqrt(2 p (1 - p)), p = mean/2, the allele-frequency scaling used by GRM / SmartPCA / GCTA).
runtime	Runtime handle from open_fmalloc(); defaults to the runtime established by init_fmalloc().

Value

A "bed" fmalloc_tensor whose decode is standardized. Monomorphic variants (zero variance) decode to `0`.

See Also

fmalloc_bed()

Examples

```
rt <- open_fmalloc(tempfile(), size_gb = 0.1)
g <- matrix(c(0L, 1L, 2L, 1L, 2L, 0L), nrow = 3, ncol = 2)
tn <- fmalloc_bed_standardize(fmalloc_bed(g, runtime = rt), runtime = rt)
round(fmalloc_tensor_materialize(tn), 3)
cleanup_fmalloc(rt)
```

`fmalloc_colVars` *Column / row variances of an fmalloc matrix*

Description

Per-column (`fmalloc_colVars`) or per-row (`fmalloc_rowVars`) sample variances, for highly-variable-feature selection and QC. Computed from the fmalloc reduction kernels (`colMeans`/`rowMeans` of `X` and `X^2`), so the result stays out-of-core-friendly and never materializes an ordinary R copy of `X`.

Usage

```
fmalloc_colVars(X)

fmalloc_rowVars(X)
```

Arguments

`X`	An fmalloc-backed numeric matrix.

Value

A numeric vector of length `ncol(X)` (`fmalloc_colVars`) or `nrow(X)` (`fmalloc_rowVars`).

`fmalloc_dosage` *Fractional genotype dosages as a 1-byte fmalloc tensor*

Description

Encodes a numeric matrix of dosages (values in `[0, 2]`, `NA` allowed) into fixed-point single-byte storage as an fmalloc_tensor of codec `"dosage"`, the continuous sibling of `fmalloc_bed()`. Samples are rows, variants are columns. A dosage `d` is stored as `round(d * 127)` (`0..254`), with `255` reserved for missing, so the resolution is `2 / 254`. This is lossy by design: it is a storage codec, eight times tighter than the doubles it decodes to, and it is the target a PLINK 2 `.pgen` dosage import re-encodes into.

Usage

```
fmalloc_dosage(x, runtime = NULL)
```

Arguments

x	A numeric matrix of dosages in [0, 2], with NA for missing.
runtime	Runtime handle from open_fmalloc(); defaults to the runtime established by init_fmalloc().

Details

Products against the tensor decode bounded column panels and contract them with BLAS, so dosages are never materialized as doubles.

Value

An fmalloc_tensor of dtype "dosage" with dim(x).

See Also

fmalloc_bed(), fmalloc_dosage_standardize()

Examples

```
rt <- open_fmalloc(tempfile(), size_gb = 0.1)
d <- matrix(c(0, 0.3, 1.7, NA, 2, 0.9), nrow = 3, ncol = 2)
tn <- fmalloc_dosage(d, runtime = rt)
round(fmalloc_tensor_materialize(tn), 2)
cleanup_fmalloc(rt)
```

---

fmalloc_dosage_standardize

Bake per-variant standardization into a dosage tensor

Description

Computes each variant's mean and standard deviation in one streaming pass and stores them in the tensor, so that every subsequent decode returns standardized, mean-imputed values: missing dosages decode to the variant mean, hence to 0 after centering. Products against the standardized tensor are therefore centered-and-scaled with no dosage ever materialized as a double and no second pass, which is what a dosage-based PCA or GRM needs. The mirror of fmalloc_bed_standardize() for continuous dosages.

Usage

```
fmalloc_dosage_standardize(x, scale = c("sd", "binomial"), runtime = NULL)
```

Arguments

x	A "dosage" fmalloc_tensor from fmalloc_dosage() (raw, not already standardized).
scale	One of "sd" (default; the sample standard deviation of the mean-imputed column, matching scale()) or "binomial" (sqrt(2 p (1 - p)), p = mean/2).
runtime	Runtime handle from open_fmalloc(); defaults to the runtime established by init_fmalloc().

Value

A "dosage" fmalloc_tensor whose decode is standardized. Monomorphic variants (zero variance) decode to 0.

See Also

fmalloc_dosage(), fmalloc_bed_standardize()

Examples

```
rt <- open_fmalloc(tempfile(), size_gb = 0.1)
d <- matrix(c(0, 0.3, 1.7, 2, 1.1, 0.9), nrow = 3, ncol = 2)
tn <- fmalloc_dosage_standardize(fmalloc_dosage(d, runtime = rt), runtime = rt)
round(fmalloc_tensor_materialize(tn), 3)
cleanup_fmalloc(rt)
```

---

fmalloc_hap_materialize

Adapt a bit-packed haplotype store to a 0/1 matrix

Description

Decodes an fmalloc_haplotypes() store back into an L x N matrix of 0L/1L integer calls, variants in rows and haplotypes in columns. The result is itself fmalloc-backed (file-mapped storage, not an R-heap copy): this is the same "decode into typed storage, not into a plain R object" discipline fmalloc_tensor_materialize() uses for the matmul codecs.

Usage

```
fmalloc_hap_materialize(x, runtime = NULL)
```

Arguments

x	An fmalloc_haplotypes object from fmalloc_haplotypes().
runtime	Runtime handle from open_fmalloc(); defaults to the runtime established by init_fmalloc().

Details

This adapter exists for consumers that require a conventional R matrix. Native HMM consumers should instead resolve `Rfmalloc_haplotypes_data()` from the installed C header and borrow the aligned locus rows directly. `kalis::CacheHaplotypes()` can consume the adapted matrix, but a kalis packed-buffer method can use the direct view without either expansion or repacking.

Value

An fmalloc-backed integer matrix of `0L`/`1L` calls with `dim(x)`.

See Also

`fmalloc_haplotypes()`

Examples

```
rt <- open_fmalloc(tempfile(), size_gb = 0.1)
h <- matrix(c(0L, 1L, 1L, 0L, 1L, 0L), nrow = 3, ncol = 2)
hap <- fmalloc_haplotypes(h, runtime = rt)
identical(fmalloc_hap_materialize(hap, runtime = rt)[], h)
cleanup_fmalloc(rt)
```

---

fmalloc_haplotypes *Phased haplotypes as a 1-bit fmalloc store*

Description

Encodes an `L x N` matrix of phased haplotype calls (`0`/`1`, variants in rows, haplotypes in columns - the convention `kalis::CacheHaplotypes()` expects for a matrix source) into fmalloc-backed, memory-mapped storage at one bit per call. The file layout is locus-major: all haplotypes at one variant form a contiguous bit row, and each row begins on a 64-byte boundary. This is the layout kalis builds internally and the access pattern Li and Stephens forward/backward kernels consume.

Usage

```
fmalloc_haplotypes(x, runtime = NULL)

create_fmalloc_haplotypes(payload, dim)

## S3 method for class 'fmalloc_haplotypes'
dim(x)

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

Arguments

x	An integer, numeric, or logical matrix of haplotype calls, values 0 or 1 only (no missing calls: phased haplotypes do not have them, and neither does kalis's own matrix input).
runtime	Runtime handle from open_fmalloc(); defaults to the runtime established by init_fmalloc().
payload	An fmalloc raw vector created by the native haplotype buffer writer.
dim	Integer dimensions c(n_variant, n_haplotype) for payload.
...	Unused.

Details

This is a SIBLING interface to the matmul fmalloc_tensor codec ABI, not an instance of it: haplotype hidden-Markov methods (Li and Stephens local- ancestry inference) are not linear algebra, so fmalloc_haplotypes() never registers a tensor codec and the resulting object never participates in %*%. It shares the same fmalloc storage substrate as fmalloc_bed() and the typed tensors, with its own encode/decode pair.

The packed body is one bit per call, asymptotically thirty-two times tighter than an integer 0/1 matrix and sixty-four times tighter than doubles. Each locus is padded to a 64-byte boundary so an HMM kernel can load donor words without repacking; that padding is visible for very small panels.

Value

An fmalloc_haplotypes object with dim(x).

See Also

fmalloc_hap_materialize() to decode back to a 0/1 matrix.

Examples

```
rt <- open_fmalloc(tempfile(), size_gb = 0.1)
h <- matrix(c(0L, 1L, 1L, 0L, 1L, 0L), nrow = 3, ncol = 2)
hap <- fmalloc_haplotypes(h, runtime = rt)
fmalloc_hap_materialize(hap, runtime = rt)
cleanup_fmalloc(rt)
```

---

fmalloc_insitu *In-place (by-reference) mutation of fmalloc vectors*

Description

Modify an fmalloc-backed atomic vector *in place*, writing straight through the backing store and deliberately bypassing R's copy-on-modify. fmalloc_set()/fmalloc_fill() are the by-reference analogue of x[i] <- value / x[] <- value; fmalloc_add()/fmalloc_sub()/ fmalloc_mul()/fmalloc_div() compute x <- x op y in place (the accumulate-into-x pattern iterative algorithms need), for numeric vectors.

Usage

```
fmalloc_set(x, i, value)

fmalloc_fill(x, value)

fmalloc_add(x, y)

fmalloc_sub(x, y)

fmalloc_mul(x, y)

fmalloc_div(x, y)
```

Arguments

x	An fmalloc-backed atomic vector (or matrix/array).
i	Positive integer (1-based) linear indices to assign.
value	For `fmalloc_set()`, a vector of length 1 (recycled) or `length(i)`. For `fmalloc_fill()`, a single scalar.
y	For the arithmetic ops, a numeric scalar (recycled) or a vector of `length(x)`. `NA`/`NaN`/`Inf` follow IEEE double arithmetic (base R semantics).

Details

On an *unshared* fmalloc vector, an ordinary `x[i] <- value` already writes in place through the ALTREP data pointer - no copy - because the file-backed storage is exposed directly and this package controls duplication. The copy that hurts happens when the vector is *shared* (`y <- x`): R then duplicates the whole payload to preserve value semantics before modifying, which is catastrophic for a larger-than-RAM or persistent vector. These functions mutate by reference regardless of sharing - they never copy, updating the durable store directly and returning the same object invisibly.

Value

`x`, invisibly, mutated in place.

Aliasing (read this)

Because there is no copy, all bindings to the same fmalloc vector observe the change. After `y <- x; fmalloc_set(x, 1, 5)`, `y[1]` is also `5` - `x` and `y` name the same backing store, whereas ordinary `x[1] <- 5` would copy `x` (leaving `y` untouched). This breaks R's usual value semantics *by design*; for a persistent runtime it is a feature (the durable data is updated). For this reason mutation is only ever done through these explicitly-named functions, never a silent `[<-` method.

Supported for fixed-width atomic vectors (logical, integer, numeric, complex, raw). Indices are 1-based linear positions (column-major for matrices/arrays).

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"))
x <- create_fmalloc_vector("numeric", 5, runtime = rt)
fmalloc_fill(x, 0)            # x[] <- 0, no copy
fmalloc_set(x, c(1, 3), 9)    # x[c(1,3)] <- 9, no copy
cleanup_fmalloc(rt)

## End(Not run)
```

`fmalloc_ld` *Banded LD (correlation) matrix as a compressed fmalloc store*

Description

Stores a banded symmetric linkage-disequilibrium (Pearson correlation) matrix in fmalloc-backed, memory-mapped storage as quantized integers. This is a SIBLING interface to the matmul fmalloc_tensor codec ABI, not an instance of it (like `fmalloc_haplotypes()`): an LD matrix is read one column's neighbours at a time by a Gibbs sampler or a ridge solve (LDpred2), never multiplied as a decoded dense `p x p` double matrix, so the store is a typed accessor with its own read API and never participates in `%*%`.

Usage

```
fmalloc_ld(i, j, x, n_variants, bits = 8L, window = 0L, runtime = NULL)

## S3 method for class 'fmalloc_ld'
dim(x)

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

Arguments

`i, j`	Integer vectors of 1-based row/column indices of the stored correlations (COO triplets). The band of column `j` is taken as the contiguous range spanning every `i` seen for that `j` (always including the diagonal); interior gaps are stored as `0`.
`x`	An `fmalloc_ld` object.
`n_variants`	Number of variants (the matrix is `n_variants` x `n_variants`).
`bits`	Quantization width, `8` (int8, the default) or `16` (int16).
`window`	Optional integer recording the build window (informational, stored in the header); `0` if unknown.
`runtime`	Runtime handle from `open_fmalloc()`; defaults to the runtime established by `init_fmalloc()`.
`...`	Unused.

Details

The variants are assumed position-sorted, so each column `j`'s in-window neighbours form a contiguous index range `[lo_j, hi_j]`. The store keeps the full symmetric band per column (both `r[j, k]` and `r[k, j]`), the diagonal is 1, and no explicit neighbour indices are stored - the row of a column's `t`-th stored value is `lo_j + t`. A per-column offset table gives O(1) random access to any column's neighbour run and a cache-friendly banded matvec.

Correlations are quantized to `bits`-wide integers: `r` in `[-1, 1]` becomes `round(r * S)` clamped to `[-S, S]`, decoded back as `q / S`, with `S = 127` for `bits = 8` (int8, resolution ~`1/127`) or `S = 32767` for `bits = 16` (int16, resolution ~`3e-5`).

`fmalloc_ld()` builds a store from `(i, j, x)` correlation triplets; RfmallocStatgen's `statgen_snp_cor()` builds one directly from a genotype tensor. Read it with `ld_ncol()`, `ld_pair()` and `ld_col()`.

Value

An `fmalloc_ld` object (a compressed, mmap-backed banded LD matrix).

See Also

`ld_ncol()`, `ld_pair()`, `ld_col()`

Examples

```
rt <- open_fmalloc(tempfile(), size_gb = 0.1)
## a 4-variant tridiagonal correlation matrix
i <- c(1, 2, 1, 2, 3, 2, 3, 4, 3, 4)
j <- c(1, 1, 2, 2, 2, 3, 3, 3, 4, 4)
x <- c(1, 0.5, 0.5, 1, 0.3, 0.3, 1, -0.2, -0.2, 1)
corr <- fmalloc_ld(i, j, x, n_variants = 4, runtime = rt)
ld_pair(corr, 1, 2)
ld_col(corr, 2)
cleanup_fmalloc(rt)
```

---

fmalloc_linalg *Matrix algebra for fmalloc-backed vectors and matrices*

Description

Provides fmalloc-aware matrix products for `%*%`, `crossprod()`, and `tcrossprod()`. Results are allocated in the runtime of the first fmalloc operand and returned as fmalloc-backed matrices.

Usage

```
## S3 method for class 'fmalloc'
x %*% y

## S3 method for class 'fmalloc'
crossprod(x, y = NULL, ...)
```

```
## S3 method for class 'fmalloc'
tcrossprod(x, y = NULL, ...)

## S3 method for class 'fmalloc'
matrixOps(x, y)
```

Arguments

x, y	Numeric, logical, or complex vectors/matrices. At least one operand must be fmalloc-backed for these methods to dispatch.
...	Unused.

Details

These methods use native C kernels to keep computations in package-managed fmalloc storage while preserving base R behavior for matrix products.

Value

An fmalloc-backed numeric or complex matrix.

---

fmalloc_matmul_ooc *Out-of-core matrix product for fmalloc matrices larger than RAM*

Description

Computes `A %*% x` where `A` is a large column-major fmalloc-backed double matrix living in the backing file and `x` is an ordinary numeric vector or matrix. `A` is consumed one contiguous column tile at a time: each tile is multiplied into the accumulator with BLAS `dgemm`, then its pages are released with `madvise(MADV_DONTNEED)`, so the resident set stays bounded by `tile_mb` rather than the size of `A`. This lets `A` exceed physical RAM.

Usage

```
fmalloc_matmul_ooc(A, x, tile_mb = 256)

fmalloc_crossprod_ooc(A, tile_mb = 256)

fmalloc_tcrossprod_ooc(A, tile_mb = 256)
```

Arguments

A	An fmalloc-backed double matrix (`m x n`).
x	A numeric vector of length `n`, or a numeric matrix (`n x k`).
tile_mb	Target resident megabytes per column tile of `A`. Larger tiles amortize BLAS overhead; smaller tiles bound peak memory more tightly. Defaults to 256.

Details

The backing storage is advised `MADV_SEQUENTIAL` so the kernel reads ahead. The result is an fmalloc-backed matrix allocated in `A`'s runtime.

`%*%` on an fmalloc matrix calls this automatically when the left operand's payload reaches `getOption("Rfmalloc.ooc_thres` (default: half of physical RAM), using `getOption("Rfmalloc.ooc_tile_mb", 256)` for the tile size; smaller products keep the in-core BLAS path. `crossprod()`/ `tcrossprod()` are not auto-routed (their output can itself exceed RAM).

Value

An fmalloc-backed double matrix (`m x k`), equal to `A %*% x`.

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"), size_gb = 8)
A <- create_fmalloc_matrix("numeric", nrow = 100000, ncol = 20000, runtime = rt)
# ... fill A ...
y <- fmalloc_matmul_ooc(A, rnorm(20000), tile_mb = 128)
cleanup_fmalloc(rt)

## End(Not run)
```

`fmalloc_pca` *Out-of-core PCA / truncated SVD for fmalloc matrices*

Description

Principal component analysis of a large, file-backed fmalloc matrix, computed from the Gram matrix: `G = X'X` via out-of-core `crossprod()`, a truncated eigendecomposition of the (small) `n x n` `G`, and the scores `X V` via out-of-core `%*%`. Every heavy step - the Gram matrix and the projection - dispatches through the pluggable matrix-multiply backend (see fmalloc_backend), so the same call runs on CPU BLAS today and on a registered GPU backend unchanged. `X` may exceed RAM: the Gram matrix and projection stream column tiles with a bounded resident set.

Usage

```
fmalloc_pca(X, k = 10L, center = TRUE)
```

Arguments

`X`	An fmalloc-backed numeric matrix (`m` observations x `n` features).
`k`	Number of principal components to return.
`center`	Logical; center the columns (applied implicitly as a rank-1 correction to the Gram matrix and scores, so `X` is never copied).

Details

This is efficient when the number of features `n` is moderate (e.g. after highly-variable-feature selection with `fmalloc_colVars()`); the `n x n` Gram matrix and its eigendecomposition are formed in memory.

Value

A list with prcomp-like elements: `sdev` (component standard deviations), `rotation` (`n x k` loadings), `x` (`m x k` scores), and `center` (the column means, or `FALSE`).

See Also

`fmalloc_colVars()`, fmalloc_backend

`fmalloc_reduction_methods` *Matrix reduction helpers for fmalloc-backed matrices*

Description

These S3 methods preserve current fmalloc behavior for matrix summary/reduction operations while returning ordinary R vectors for small results.

Usage

```
rowSums(x, na.rm = FALSE, dims = 1L)

colSums(x, na.rm = FALSE, dims = 1L)

rowMeans(x, na.rm = FALSE, dims = 1L)

colMeans(x, na.rm = FALSE, dims = 1L)
```

Arguments

`x`	A matrix-like object.
`na.rm`	Logical scalar controlling NA removal.
`dims`	Numeric scalar for dimensions.

Details

These implementations keep managed execution for 2D `fmalloc` matrices with `dims = 1L`. For unsupported shapes or `dims` values (for example, non-2D arrays or `dims != 1L`), the methods warn and delegate to the base R implementations (`base::rowSums`, `base::colSums`, `base::rowMeans`, and `base::colMeans`).

Value

The reduction result, as either an ordinary R object or a `fmalloc` vector when result length exceeds `getOption("Rfmalloc.reduce_result_length", 1e6)`.

`fmalloc_storage_advise` *Give the pager an access hint for typed storage*

Description

Applies a best-effort access hint to an fmalloc tensor payload or borrowed storage view. `"sequential"` asks the pager to read ahead, `"willneed"` asks it to prefetch, and `"dontneed"` releases fully covered pages after a phase has consumed them. Unsupported platforms treat the hint as a no-op. The function never changes the tensor bytes.

Usage

```
fmalloc_storage_advise(
  x,
  advice = c("sequential", "willneed", "dontneed"),
  offset = 0,
  nbytes = NULL
)
```

Arguments

`x`	An `fmalloc_tensor` or its raw fmalloc payload.
`advice`	One of `"sequential"`, `"willneed"`, or `"dontneed"`.
`offset`	Byte offset into the payload.
`nbytes`	Number of bytes to advise. `NULL` covers the rest of the payload.

Details

This is deliberately a storage primitive rather than an LLM policy. A graph scheduler, HMM, or out-of-core matrix algorithm can express its own access phases over the same mapped spans.

Value

`x`, invisibly.

fmalloc_sync *Flush an fmalloc runtime's backing store to disk*

Description

Writes to an fmalloc runtime (including in-place mutations via `fmalloc_set()` and friends) land in the OS page cache of the `MAP_SHARED` backing file. They survive a normal process exit, but until the kernel writes dirty pages back - which it does asynchronously - a crash or power loss can lose unsynced data, with no atomicity. `fmalloc_sync()` forces the durability barrier with `msync()` (and `fsync()`), so persistent data is on disk when it returns.

Usage

```
fmalloc_sync(runtime = NULL, wait = TRUE)
```

Arguments

`runtime`	Runtime handle from `open_fmalloc()`; defaults to the runtime established by `init_fmalloc()`.
`wait`	If `TRUE` (default), block until the flush completes (`MS_SYNC`); if `FALSE`, schedule an asynchronous flush (`MS_ASYNC`).

Details

This matters only for `persistent` runtimes where durability across an unclean shutdown is required; `scratch` runtimes are ephemeral by design. On platforms without `msync` (e.g. the Windows/Rtools toolchain) this is a no-op and returns 0, relying on OS writeback.

Value

The number of bytes flushed, invisibly.

fmalloc_tensor *Typed fmalloc tensors*

Description

A typed fmalloc tensor is an fmalloc raw vector holding a matrix payload in a foreign element encoding (`"f32"`, `"f16"`, `"bf16"`, or any codec registered by another package through the `Rfmalloc_register_tensor_codec` C-callable), plus dimension and dtype tags. Matrix products against dense double operands decode the payload in bounded, block-aligned column panels that are streamed through BLAS `dgemm`, so the double representation of the full tensor is never materialized at once.

Usage

```
fmalloc_tensor_codecs()

create_fmalloc_tensor(payload, dtype, dim)

as_fmalloc_tensor(x, dtype = "alp", runtime = NULL)

fmalloc_tensor_dtype(x)

fmalloc_tensor_materialize(x)

## S3 method for class 'fmalloc_tensor'
dim(x)

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

## S3 method for class 'fmalloc_tensor'
x %*% y

## S3 method for class 'fmalloc_tensor'
matrixOps(x, y)

## S3 method for class 'fmalloc_tensor'
crossprod(x, y = NULL, ...)

## S3 method for class 'fmalloc_tensor'
tcrossprod(x, y = NULL, ...)
```

Arguments

payload	An fmalloc raw vector holding the encoded payload in column-major order (first dimension fastest).
dtype	Codec name, e.g. "f32", "f16", "bf16"; for as_fmalloc_tensor(), "alp" or "sparse".
dim	Integer dimensions of the decoded tensor (any rank). Storage and fmalloc_tensor_materialize() handle any number of dimensions; the matrix products (%*%, crossprod(), tcrossprod()) require exactly 2.
x	An fmalloc_tensor object (or, in %*%, a dense operand).
runtime	Optional runtime handle from open_fmalloc(); defaults to the runtime established by init_fmalloc().
...	Unused.
y	The other matrix product operand.

Details

create_fmalloc_tensor() tags an existing fmalloc raw payload. as_fmalloc_tensor() com-

presses a double vector/matrix into fmalloc storage with `dtype = "sparse"` (stores only the nonzeros of each chunk, for mostly-zero data such as single-cell counts) or the builtin, lossless `"alp"` codec (Afroozeh et al., doi:10.1145/3626717; scalar core adapted from the MIT-licensed zap implementation, see `inst/COPYRIGHTS`), storing decimal-scaled doubles as bit-packed integers in independently decodable 1024-value chunks with exact-value patches and a raw escape hatch for incompressible chunks. `fmalloc_tensor_materialize()` decodes the whole tensor into an fmalloc double matrix. `fmalloc_tensor_codecs()` lists registered codec names, and `fmalloc_tensor_dtype()` returns a tensor's dtype tag.

When a tensor's compressed payload reaches `getOption("Rfmalloc.ooc_threshold_gb")`, its matrix products stream out-of-core: each column panel's source pages are released after decoding (for fixed-geometry codecs), so a tensor whose decoded `f64` form exceeds RAM multiplies with a bounded resident set.

Value

`create_fmalloc_tensor()` returns an `fmalloc_tensor`. `fmalloc_tensor_materialize()` and the matrix products return fmalloc-backed double matrices. `fmalloc_tensor_codecs()` returns a character vector.

`init_fmalloc` *Initialize fmalloc*

Description

Compatibility wrapper that opens an fmalloc runtime and installs it as the package default runtime used by `create_fmalloc_vector()` when no explicit runtime is supplied.

Usage

```
init_fmalloc(filepath, size_gb = NULL, mode = c("persistent", "scratch"))
```

Arguments

`filepath`	Character string specifying the file path for fmalloc data.
`size_gb`	Numeric value specifying the size of the backing file in GB (optional). If not specified, uses the package default size for new files or the existing file size.
`mode`	Runtime mode. `"persistent"` keeps committed vector payloads in the backing file and serializes fixed-width atomic vectors by reference. `"scratch"` uses the backing file as a large temporary allocation arena and serializes vectors by value.

Details

For new code, prefer `open_fmalloc()` and pass the returned runtime handle to `create_fmalloc_vector()`.

Value

Logical indicating whether the file was newly initialized.

Examples

```
## Not run:
alloc_file <- tempfile(fileext = ".bin")
init_fmalloc(alloc_file)
v <- create_fmalloc_vector("integer", 1000)
cleanup_fmalloc()
unlink(alloc_file)

## End(Not run)
```

`ld_col` *Neighbour run of one column of a banded LD store*

Description

Returns column `j`'s contiguous band of correlations: the decoded values for rows `lo..hi` (the window around variant `j`), where `hi - lo + 1` is the band length. This is the O(1) per-column access the LDpred2 banded matvec rides.

Usage

```
ld_col(store, j)
```

Arguments

`store`	An fmalloc_ld object.
`j`	1-based column index.

Value

A list with `lo` and `hi` (1-based inclusive row bounds of the band) and `x` (the decoded correlations for rows `lo:hi`, length `hi - lo + 1`).

See Also

`fmalloc_ld()`, `ld_pair()`, `ld_ncol()`

ld_ncol *Number of variants in a banded LD store*

Description

Number of variants in a banded LD store

Usage

```
ld_ncol(store)
```

Arguments

store	An fmalloc_ld object.

Value

The number of variants (columns == rows) as an integer.

See Also

fmalloc_ld(), ld_pair(), ld_col()

ld_pair *Correlation between two variants of a banded LD store*

Description

Returns r[i, j] from a banded LD store. Pairs outside the stored band (too far apart to be in each other's window) return 0.

Usage

```
ld_pair(store, i, j)
```

Arguments

store	An fmalloc_ld object.
i, j	1-based variant indices.

Value

The (quantized) correlation r[i, j], or 0 if the pair is outside the band.

See Also

fmalloc_ld(), ld_col(), ld_ncol()

`list_fmalloc_allocations` *List Persistent fmalloc Allocations*

Description

Returns the in-file allocation catalog for a persistent fmalloc runtime. The catalog is stored in the backing file and records physical allocation metadata used to validate serialized persistent references.

Usage

```
list_fmalloc_allocations(runtime = NULL)
```

Arguments

`runtime`	Optional runtime handle returned by `open_fmalloc()`. If not supplied, the default runtime established by `init_fmalloc()` is used.

Details

For successful recovery, look at the `state` column:

- `"committed"`: valid serialized payload exists for that record;
- `"tombstone"`: the payload has been destroyed and is non-recoverable unless the runtime remains open and referenced directly by an existing SEXP;
- other transient states are internal and are generally not expected.

`recoverable` indicates whether the record can be reopened via serialized reference metadata. `payload_offset == 0` or `payload_nbytes == 0` generally indicates a non-payload entry.

Value

A data frame with one row per catalog record and columns describing the catalog record offset, generation, state, vector type, length, payload offset, payload byte size, flags, and whether the record is recoverable by reference serialization.

Examples

```
## Not run:
rt <- open_fmalloc(tempfile(fileext = ".bin"))
v <- create_fmalloc_vector("integer", 10, runtime = rt)
list_fmalloc_allocations(rt)
cleanup_fmalloc(rt)

## End(Not run)
```

`open_fmalloc` *Open an fmalloc Runtime*

Description

Opens a file-backed fmalloc runtime and returns an external-pointer handle. Multiple handles to the same path share the same underlying runtime within a process. Runtime mode mismatches are rejected for an already-open path. Runtime mode controls whether vector payloads are durable persistent allocations or scratch allocations that can be returned to fmalloc when their ALTREP handles are garbage-collected.

Usage

```
open_fmalloc(filepath, size_gb = NULL, mode = c("persistent", "scratch"))
```

Arguments

`filepath`	Character string specifying the file path for fmalloc data.
`size_gb`	Numeric value specifying the size of the backing file in GB (optional). If not specified, uses the package default size for new files or the existing file size.
`mode`	Runtime mode. `"persistent"` keeps committed vector payloads in the backing file and serializes fixed-width atomic vectors by reference. `"scratch"` uses the backing file as a large temporary allocation arena and serializes vectors by value.

Value

An external pointer of class `fmalloc_runtime`.

Index