Package: fmristore (via r-universe)

June 10, 2026

Type Package

Title Efficient Storage of fMRI Data

Version 0.1.0

Description Provides efficient storage and retrieval of functional magnetic resonance imaging (fMRI) data using HDF5 format. The package offers S4 classes and methods for working with dense and sparse representations of 4D neuroimaging data, latent space decompositions, and spatially clustered voxel time series. It includes memory-efficient access patterns for large datasets and integrates with the neuroim2 package for standard neuroimaging data structures.

License GPL (>= 3)

URL https://github.com/bbuchsbaum/fmristore, https://bbuchsbaum.github.io/fmristore/

BugReports https://github.com/bbuchsbaum/fmristore/issues

Encoding UTF-8

Language en-US

LazyData false

RoxygenNote 7.3.3

Imports assertthat, cli, fmrilatent, hdf5r, lifecycle, Matrix, methods, neuroim2, withr

Suggests albersdown, corrplot, ggplot2, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

Collate 'all_class.R' 'all_generic.R' 'assertions.R' 'cluster_array.R' 'cluster_experiment.R' 'constants.R' 'constructors.R' 'fmristore-package.R' 'globals.R' 'h5_utils.R' 'h5neurovec.R' 'h5neurovecseq.R' 'h5neurovol.R' 'io_h5_generic.R' 'io_h5_helpers.R' 'io_write_h5.R' 'labeled_vec.R'

'latent_vec.R' 'memory_utils.R' 'neurovecseq_h5.R'
'read_dataset_methods.R' 'write_cluster_result.R'
'write_dataset_methods.R' 'write_parcellated_scan_h5.R'
'zzz_example_helpers.R'

Remotes bbuchsbaum/neuroim2, bbuchsbaum/fmrilatent, bbuchsbaum/albersdown

Config/Needs/website albersdown

Config/pak/sysreqs make libhdf5-dev libicu-dev

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

Date/Publication 2026-03-29 15:11:58 UTC

RemoteUrl https://github.com/bbuchsbaum/fmristore

RemoteRef HEAD

RemoteSha 0b834048f20f6a9cfc3c0bc36fa171ca71558359

Contents

[,H5NeuroVol,ANY,ANY,ANY-method . . . 4
[,H5NeuroVol,numeric,missing,ANY-method . . . 4
[,H5ParcellatedScan,ANY,ANY,ANY-method . . . 5
[,H5ParcellatedScan,ANY,missing,ANY-method . . . 6
[,LabeledVolumeSet,ANY,ANY,ANY-method . . . 7
$,H5ParcellatedMultiScan-method . . . 8
as.array.LatentNeuroVec . . . 8
as.data.frame . . . 9
as.matrix . . . 10
as_h5 . . . 10
as_h5,NeuroVol-method . . . 14
basis . . . 15
close . . . 16
cluster_metadata . . . 18
cluster_metadata,H5ParcellatedMultiScan-method . . . 19
clusters . . . 20
configure_memory_warnings . . . 21
detect_h5_type . . . 22
get_memory_settings . . . 23
H5_ATTRS . . . 24
H5_DSETS . . . 24
H5_PATHS . . . 25
h5file . . . 25
h5file,H5ParcellatedArray-method . . . 26
H5NeuroVec . . . 27
H5NeuroVec-class . . . 29
H5NeuroVecSeq . . . 30
H5NeuroVecSeq-class . . . 31
H5NeuroVol . . . 31

H5NeuroVol-class . . . 32
H5ParcellatedMultiScan . . . 34
H5ParcellatedMultiScan-class . . . 36
H5ParcellatedScan . . . 36
H5ParcellatedScan-class . . . 38
H5ParcellatedScanSummary . . . 38
H5ParcellatedScanSummary-class . . . 40
LabeledVolumeSet-class . . . 41
LatentNeuroVec . . . 42
LatentNeuroVec-io . . . 43
loadings . . . 43
make_run_full . . . 44
make_run_summary . . . 46
map . . . 47
mask . . . 48
mask,H5ParcellatedArray-method . . . 49
matrix_concat . . . 50
n_scans . . . 51
n_scans,H5ParcellatedMultiScan-method . . . 53
neurovecseq_to_h5 . . . 53
offset . . . 55
read_dataset . . . 56
read_dataset,character-method . . . 57
read_labeled_vec . . . 58
scan_metadata . . . 59
scan_metadata,H5ParcellatedMultiScan-method . . . 60
scan_names . . . 61
scan_names,H5ParcellatedMultiScan-method . . . 62
series . . . 63
series_concat . . . 64
show,H5NeuroVec-method . . . 66
show,H5NeuroVol-method . . . 66
show,H5ParcellatedScan-method . . . 67
show,LabeledVolumeSet-method . . . 67
validate_latent_file . . . 68
write_cluster_result . . . 69
write_dataset . . . 71
write_labeled_vec . . . 74
write_parcellated_experiment_h5 . . . 76
write_parcellated_scan_h5 . . . 80
write_vec,LatentNeuroVec,character,missing,missing-method . . . 81

Index **83**

`[,H5NeuroVol,ANY,ANY,ANY-method`

3D bracket subsetting for H5NeuroVol (handles partial arguments)

Description

Allows `h5vol[i, j, k]` where each of `i,j,k` may be missing. Missing arguments default to the entire range in that dimension. Zero-length arguments immediately yield an empty array of the correct shape.

Usage

```
## S4 method for signature 'H5NeuroVol,ANY,ANY,ANY'
x[i, j, k, ..., drop = TRUE]
```

Arguments

`x`	An `H5NeuroVol` instance
`i, j, k`	Numeric (or integer) index vectors for each dimension. If missing, we take the full range in that dimension.
`...`	Unused
`drop`	Logical: whether to drop dimensions of size 1. Default `TRUE`.

Value

A numeric `array` of shape `c(length(i), length(j), length(k))`, or fewer dims if `drop=TRUE`.

`[,H5NeuroVol,numeric,missing,ANY-method`

Single-vector indexing for H5NeuroVol

Description

Handles `h5vol[i]` where only `i` is provided. If all indices are within the first dimension range, treats as first-dimension slicing. Otherwise, treats as linear indexing into the flattened volume.

Usage

```
## S4 method for signature 'H5NeuroVol,numeric,missing,ANY'
x[i, j, ..., drop = TRUE]
```

Arguments

x	An H5NeuroVol instance
i	Numeric index vector
j	Missing
...	Unused
drop	Logical: whether to drop dimensions of size 1. Default TRUE.

Value

Values from the volume (either as array slice or linear-indexed values)

[,H5ParcellatedScan,ANY,ANY,ANY-method

Subset an H5ParcellatedScan Object

Description

Extracts data from an H5ParcellatedScan object using array-like indexing. Handles both coordinate-based and mask-index-based subsetting.

Usage

```
## S4 method for signature 'H5ParcellatedScan,ANY,ANY,ANY'
x[i, j, k, l, ..., drop = TRUE]
```

Arguments

x	An H5ParcellatedScan object.
i	Row index (x-coordinate or mask index).
j	Column index (y-coordinate).
k	Slice index (z-coordinate).
l	Time index.
...	Not used.
drop	Logical. If TRUE, the result is coerced to the lowest possible dimension.

Value

An array or vector containing the subset of data.

See Also

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), series(), show,H5ParcellatedScan-method

`[,H5ParcellatedScan,ANY,missing,ANY-method`
Extract data from H5ParcellatedScan with missing j parameter

Description

Extracts data from an `H5ParcellatedScan` object when only the first index is provided. Can handle mask indices, coordinate matrices, or pass through to coordinate-based indexing.

Extracts an individual 'H5NeuroVec' from an 'H5NeuroVecSeq' object using double bracket notation.

Usage

```
## S4 method for signature 'H5ParcellatedScan,ANY,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'H5ParcellatedScanSummary,ANY,ANY,ANY'
x[i, j, k, l, ..., drop = TRUE]

## S4 method for signature 'H5ParcellatedScanSummary,ANY,missing,ANY'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'H5NeuroVec,numeric,numeric,ANY'
x[i, j, k, l, ..., drop = TRUE]

## S4 method for signature 'H5NeuroVecSeq,ANY'
x[[i, j, ...]]

## S4 method for signature 'LabeledVolumeSet,numeric'
x[[i, j, ...]]

## S4 method for signature 'LabeledVolumeSet,character'
x[[i, j, ...]]
```

Arguments

`x`	An 'H5NeuroVecSeq' object.
`i`	Index (numeric or character) specifying which scan to extract.
`j`	Not used.
`...`	Not used.
`drop`	Logical. If `TRUE`, the result is coerced to the lowest possible dimension
`k`	Slice index (z-coordinate) - optional, passed via ...
`l`	Time index - optional, passed via ...

Value

An array or vector containing the subset of data

An 'H5NeuroVec' object for the requested scan.

See Also

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), series(), show,H5ParcellatedScan-method

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), series(), show,H5ParcellatedScan-method

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), series(), show,H5ParcellatedScan-method

---

[,LabeledVolumeSet,ANY,ANY,ANY-method

4D Array-like subsetting for LabeledVolumeSet

Description

4D Array-like subsetting for LabeledVolumeSet

Usage

```
## S4 method for signature 'LabeledVolumeSet,ANY,ANY,ANY'
x[i, j, k, l, ..., drop = TRUE]
```

Arguments

x	A LabeledVolumeSet object.
i	Numeric indices for the 1st dimension (x).
j	Numeric indices for the 2nd dimension (y).
k	Numeric indices for the 3rd dimension (z).
l	Numeric indices for the 4th dimension (label).
...	Ignored.
drop	Logical, whether to drop singleton dimensions.

Value

An R array after subsetting, or a lower-dimensional array if drop=TRUE.

`$,H5ParcellatedMultiScan-method`
Accessor Methods for H5ParcellatedMultiScan Access Slots/Properties using '$'

Description

Provides convenient access to shared properties like 'mask', 'clusters', and the underlying HDF5 file object ('obj') by retrieving them from the first run object stored within the experiment. Also provides access to the experiment's own slots ('runs', 'scan_metadata', 'cluster_metadata').

Usage

```
## S4 method for signature 'H5ParcellatedMultiScan'
x$name
```

Arguments

`x`	An 'H5ParcellatedMultiScan' object.
`name`	The name of the property or slot to access ('mask', 'clusters', 'obj', 'runs', 'scan_metadata', 'cluster_metadata').

Value

The requested object or value.

See Also

Other H5Parcellated: `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`, `close()`, `cluster_metadata,H5ParcellatedMultiScan-method`, `clusters()`, `h5file,H5ParcellatedArray-method`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

`as.array.LatentNeuroVec`
Convert LatentNeuroVec to Array

Description

Converts a 'LatentNeuroVec' to a standard R 4D array by reconstructing the full data from the latent representation.

Usage

```
## S3 method for class 'LatentNeuroVec'
as.array(x, ...)
```

Arguments

x	A 'LatentNeuroVec' object to convert.
...	Not used.

Details

This method reconstructs the full 4D array from the latent representation, which can be memory-intensive for large datasets. It calculates:

$$array[,,,t] = Map(basis[t,] * t(loadings)) + offset$$

for each time point, where values outside the mask are set to zero.

Value

A 4D array representing the full reconstructed data.

---

as.data.frame *Convert to Data Frame*

Description

These methods convert 'fmristore' specific objects to data frames, implementing the S4 generic function `as.data.frame` from the 'base' package.

Usage

```
as.data.frame(x, row.names = NULL, optional = FALSE, ...)

## S4 method for signature 'H5ParcellatedScanSummary'
as.data.frame(x, row.names = NULL, optional = FALSE, ...)
```

Arguments

x	An object for which a 'as.data.frame' method is defined.
row.names	A character vector giving the row names for the data frame, or NULL.
optional	Logical. If TRUE, setting row names and converting column names is optional.
...	Additional arguments passed to the underlying as.data.frame methods.

See Also

Other H5Cluster: `H5ParcellatedMultiScan-class`, `[,H5ParcellatedScan,ANY,ANY,ANY-method`, `[,H5ParcellatedScan,ANY,missing,ANY-method`, `as.matrix()`, `dim()`, `linear_access-methods`, `make_run_full()`, `make_run_summary()`, `series()`, `show,H5ParcellatedScan-method`

as.matrix *Convert to Matrix*

Description

These methods convert ‘fmristore‘ specific objects to matrices, implementing the S4 generic function `as.matrix` from the ‘base‘ package.

Usage

```
as.matrix(x, ...)

## S4 method for signature 'H5ParcellatedScanSummary'
as.matrix(x)
```

Arguments

x	An object for which a ‘as.matrix‘ method is defined.
...	Additional arguments passed to specific as.matrix methods.

See Also

Other H5Cluster: `H5ParcellatedMultiScan-class`, `[,H5ParcellatedScan,ANY,ANY,ANY-method`, `[,H5ParcellatedScan,ANY,missing,ANY-method`, `as.data.frame()`, `dim()`, `linear_access-methods`, `make_run_full()`, `make_run_summary()`, `series()`, `show,H5ParcellatedScan-method`

as_h5 *Generic function to convert R objects to HDF5 format*

Description

A generic function for converting various types of R objects to HDF5 format, providing a standardized interface for serialization to HDF5.

Usage

```
as_h5(object, file, ...)

## S4 method for signature 'NeuroVec'
as_h5(
  object,
  file = NULL,
  data_type = "FLOAT",
  chunk_dim = c(4, 4, 4, dim(object)[4]),
  compression = 6
```

```
)

## S4 method for signature 'LatentNeuroVec'
as_h5(object, file = NULL, data_type = "FLOAT", compression = 6)

## S4 method for signature 'LabeledVolumeSet'
as_h5(
  object,
  file,
  mask,
  labels,
  compression = 4,
  dtype = hdf5r::h5types$H5T_NATIVE_DOUBLE,
  chunk_size = 1024,
  header_values = list()
)

## S4 method for signature 'ClusteredNeuroVec'
as_h5(
  object,
  file = NULL,
  scan_name = "scan_001",
  as_multiscan = FALSE,
  compression = 4,
  ...
)

## S4 method for signature 'NeuroVecSeq'
as_h5(object, file = NULL, ...)
```

Arguments

object	The R object to convert to HDF5 (e.g., a NeuroVol or NeuroVec).
file	The path to the HDF5 file to create or modify.
...	Additional arguments specific to the particular method (see Details).
data_type	For NeuroVec/LatentNeuroVec methods: Storage type (e.g., "FLOAT"). Default "FLOAT"
chunk_dim	For NeuroVec method: Chunk dimensions. Default depends on input dimensions
compression	For all methods: Integer compression level [0..9]. Default varies by method
mask	For LabeledVolumeSet method: The mask to use (LogicalNeuroVol)
labels	For LabeledVolumeSet method: Character vector of labels
dtype	For LabeledVolumeSet method: HDF5 data type for values. Default H5T_NATIVE_DOUBLE
chunk_size	For LabeledVolumeSet method: Integer chunk size for HDF5. Default 1024
header_values	For LabeledVolumeSet method: List of additional header values
scan_name	For ClusteredNeuroVec method: Character name for this scan (default "scan_001")
as_multiscan	For ClusteredNeuroVec method: If TRUE, creates multi-scan container

Value

An object representing the HDF5 storage, typically of a class corresponding to the input type (e.g., `H5NeuroVol` for `NeuroVol` input, `H5NeuroVec` for `NeuroVec` input).

Methods

`signature(object = "NeuroVec")` Creates an HDF5 file from a 4D NeuroVec object. Additional parameters:

`data_type` Storage type (e.g., "FLOAT"). Default "FLOAT"

`chunk_dim` Chunk dimensions. Default depends on input dimensions

`compression` Integer [0..9], default 6

Returns an H5NeuroVec referencing the new HDF5 file.

`signature(object = "LatentNeuroVec")` Saves a LatentNeuroVec to an HDF5 file in BasisEmbeddingSpec format. Additional parameters:

`data_type` Storage type (e.g., "FLOAT"). Default "FLOAT"

`compression` Integer [1..9], default 6

Returns an HDF5 file object.

`signature(object = "LabeledVolume")` Saves a LabeledVolume to an HDF5 file. Additional parameters:

`mask` The mask to use (LogicalNeuroVol)

`labels` Character vector of labels

`compression` Integer [0..9], default 4

`dtype` HDF5 data type for values. Default H5T_NATIVE_DOUBLE

`chunk_size` Integer chunk size for HDF5, default 1024

`header_values` List of additional header values

Returns an HDF5 file object.

`signature(object = "NeuroVecSeq")` Writes a sequence of NeuroVec objects (multiple 4D scans) to a single HDF5 file. All NeuroVec objects must have the same spatial dimensions. Additional parameters:

`scan_names` Optional character vector of scan names. If NULL, uses "scan_1", "scan_2", etc.

`data_type` Character string: "FLOAT" (default), "DOUBLE", or "INT"

`chunk_dim` Numeric vector specifying chunk sizes. If NULL, uses time-optimized chunking

`compression` Integer [0..9], default 6

`scan_metadata` Optional named list of metadata lists, one per scan

Returns the file path of the created HDF5 file.

Examples

```
## Not run:
# Example 1: NeuroVec (DenseNeuroVec) to HDF5
# Ensure helper function is available and as_h5 exists
# if (!is.null(fmristore:::create_minimal_DenseNeuroVec) &&
```

```
#     exists("as_h5", where = "package:fmristore")) {

dvec <- fmristore:::create_minimal_DenseNeuroVec(dims = c(3L,3L,2L,4L))
temp_h5_file <- tempfile(fileext = ".h5")
h5_obj <- NULL

tryCatch({
    # Convert DenseNeuroVec to an HDF5 file and get an H5NeuroVec object back
    h5_obj <- as_h5(dvec, file = temp_h5_file,
                    data_type = "FLOAT",
                    chunk_dim = c(2, 2, 2, 4),
                    compression = 4)

    print(h5_obj) # Should be an H5NeuroVec

}, error = function(e) {
    message("as_h5 NeuroVec example failed: ", e$message)
}, finally = {
    if (!is.null(h5_obj)) try(close(h5_obj), silent = TRUE)
    if (file.exists(temp_h5_file)) {
      unlink(temp_h5_file)
    }
})
# }

# Example 2: LatentNeuroVec to HDF5
# if (!is.null(fmristore:::create_minimal_LatentNeuroVec) &&
#     exists("as_h5", where = "package:fmristore")) {

lnv <- fmristore:::create_minimal_LatentNeuroVec(
    space_dims = c(4L, 4L, 2L),
    n_time = 6L,
    n_comp = 2L
)
temp_h5_file_lnv <- tempfile(fileext = ".h5") # Use a different temp file name
h5_obj_lnv <- NULL # Use a different object name

tryCatch({
    # Convert LatentNeuroVec to HDF5
    h5_obj_lnv <- as_h5(lnv, file = temp_h5_file_lnv, compression = 4)

    # File should exist and h5_obj_lnv should be a valid H5File object
    if (file.exists(temp_h5_file_lnv)) {
      print("LatentNeuroVec HDF5 file created successfully: ", temp_h5_file_lnv)
    }

}, error = function(e) {
    message("as_h5 LatentNeuroVec example failed: ", e$message)
}, finally = {
    if (!is.null(h5_obj_lnv) && inherits(h5_obj_lnv, "H5File") && h5_obj_lnv$is_valid) {
      try(h5_obj_lnv$close_all(), silent = TRUE)
    }
    if (file.exists(temp_h5_file_lnv)) {
```

```
      unlink(temp_h5_file_lnv)
    }
})
# }

# Example 3: Clustered dataset (list) to HDF5
# (This requires more complex setup, so we use a simplified theoretical example)
\dontrun{
  # Removed: if (requireNamespace("neuroim2", quietly = TRUE) && ...

  # In practice, you would:
  # 1. Create a list of NeuroVec/DenseNeuroVec objects (scan data)
  # 2. Create a LogicalNeuroVol mask
  # 3. Create a ClusteredNeuroVol defining clusters
  # 4. Define scan names and metadata
  # 5. Call as_h5() with these components

  # Note: This example is simplified for documentation and won't execute
  # as the actual clustered dataset structure is more complex

  message("Example usage for clustered dataset (as_h5 for list method):")
  message("  # vecs_list <- ... list of NeuroVec objects ...")
  message("  # mask_obj <- fmristore:::create_minimal_LogicalNeuroVol(...)")
  message("  # clusters_obj <- fmristore:::create_minimal_ClusteredNeuroVol(...)")
  message("  # scan_meta <- list(list(TR=2), list(TR=2))")
  message("  # h5_clust_obj <- as_h5(vecs_list, file = tempfile(fileext=\".h5\"), ")
  message("  #                       scan_names = c('run1', 'run2'), ")
  message("  #                       mask = mask_obj, clusters = clusters_obj, ")
  message("  #                       scan_metadata = scan_meta)")
  message("  # print(h5_clust_obj)")
  message("  # if (!is.null(h5_clust_obj)) try(close(h5_clust_obj), silent=TRUE)")
  message("  # if (file.exists(attr(h5_clust_obj, \"filepath\"))) ")
  message("  #     unlink(attr(h5_clust_obj, \"filepath\"))")

}

## End(Not run)
```

---

as_h5,NeuroVol-method *Convert a NeuroVol to HDF5 Format (as_h5 Method)*

Description

Saves a NeuroVol to an HDF5 file with minimal necessary metadata to reconstruct an H5NeuroVol.

Usage

```
## S4 method for signature 'NeuroVol'
as_h5(
```

```
  object,
  file = NULL,
  data_type = "FLOAT",
  chunk_dim = NULL,
  compression = 6
)
```

Arguments

object	A NeuroVol object (3D)
file	Path to the output file (if NULL, uses tempfile)
data_type	Character: "FLOAT", "DOUBLE", "INT", etc.
chunk_dim	Numeric vector specifying chunk sizes
compression	Integer [1..9], default 6

Value

A new H5NeuroVol referencing the written file. The returned object contains an open read-mode HDF5 handle. **Important:** The user is responsible for closing this handle using close() on the returned object when finished.

---

basis *Get the basis matrix (temporal components)*

Description

Get the basis matrix (temporal components)

Usage

```
basis(x, ...)
```

Arguments

x	An object, likely a LatentNeuroVec or similar
...	Additional arguments

Value

The basis matrix (typically time x components)

Examples

```
## Not run:
# For LatentNeuroVec:
if (!is.null(fmristore:::create_minimal_LatentNeuroVec)) {
  lnv <- NULL
  tryCatch({
    lnv <- fmristore:::create_minimal_LatentNeuroVec(
      space_dims = c(4L, 4L, 2L),
      n_time = 10L,
      n_comp = 3L
    )
    b_matrix <- basis(lnv)
    print(dim(b_matrix)) # Should be n_time x n_comp (e.g., 10x3)
  }, error = function(e) {
    message("basis example for LatentNeuroVec failed: ", e$message)
  })
} else {
  message("Skipping basis example for LatentNeuroVec: helper not available.")
}

## End(Not run)
```

`close` *Close an Object*

Description

Close connections or release resources associated with an object. This generic is used for objects that manage external resources like HDF5 file handles.

This method manually closes the HDF5 file handle stored within the H5NeuroVec object. It uses the `close_h5_safely` helper.

This method manually closes the HDF5 file handle stored within the H5NeuroVol object. It uses the `safe_h5_close` helper.

This method manually closes the HDF5 file handle stored within the LabeledVolumeSet object. It uses the `close_h5_safely` helper to ensure the handle is valid before attempting to close.

Usage

```
close(con, ...)

## S4 method for signature 'H5ParcellatedMultiScan'
close(con, ...)

## S4 method for signature 'H5ParcellatedScanSummary'
close(con, ...)
```

```
## S4 method for signature 'H5ParcellatedScan'
close(con, ...)

## S4 method for signature 'H5NeuroVec'
close(con, ...)

## S4 method for signature 'H5NeuroVecSeq'
close(con, ...)

## S4 method for signature 'H5NeuroVol'
close(con, ...)

## S4 method for signature 'LabeledVolumeSet'
close(con, ...)
```

Arguments

con	A LabeledVolumeSet object.
...	Additional arguments (ignored).

Value

Usually returns NULL invisibly

NULL invisibly

Invisibly returns NULL.

Invisibly returns NULL.

Invisibly returns NULL.

See Also

Other H5Parcellated: $,H5ParcellatedMultiScan-method, H5ParcellatedArray-class, H5ParcellatedMultiScan, H5ParcellatedScan-class, H5ParcellatedScanSummary-class, cluster_metadata,H5ParcellatedMultiScan-metho clusters(), h5file,H5ParcellatedArray-method, mask,H5ParcellatedArray-method, matrix_concat(), n_scans,H5ParcellatedMultiScan-method, scan_metadata,H5ParcellatedMultiScan-method, scan_names,H5ParcellatedMultiScan-method, series_concat()

Examples

```
## Not run:
# Example with LabeledVolumeSet
if (!is.null(fmristore:::create_minimal_h5_for_LabeledVolumeSet)) {
  temp_file <- fmristore:::create_minimal_h5_for_LabeledVolumeSet()
  lvs <- read_labeled_vec(temp_file)
  # Use the object...
  close(lvs) # Close the HDF5 file handle
  unlink(temp_file)
}

## End(Not run)
```

`cluster_metadata` *Get cluster metadata*

Description

This generic returns any available metadata associated with clusters in an object.

Usage

```
cluster_metadata(x)
```

Arguments

`x`	The object from which to retrieve the cluster metadata

Value

A data frame or other structure containing metadata for each cluster

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("hdf5r", quietly = TRUE) &&
    exists("H5ParcellatedMultiScan", where = "package:fmristore") &&
    exists("cluster_metadata", where = "package:fmristore") &&
    !is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {

  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    # Create a minimal H5ParcellatedMultiScan
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan()
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

    # Get the cluster metadata
    c_meta <- cluster_metadata(exp_obj)
    print(c_meta)
    # The helper currently doesn't add rich cluster_metadata,
    # so this is likely an empty data.frame or one with default cluster names/IDs.

  }, error = function(e) {
    message("cluster_metadata example failed: ", e$message)
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
```

```
  })
} else {
  message("Skipping cluster_metadata example: dependencies or helper not available.")
}

## End(Not run)
```

`cluster_metadata,H5ParcellatedMultiScan-method`
Get cluster metadata

Description

Get cluster metadata

Usage

```
## S4 method for signature 'H5ParcellatedMultiScan'
cluster_metadata(x)
```

Arguments

x	H5ParcellatedMultiScan object

Value

Data frame of cluster metadata.

See Also

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`, `close()`, `clusters()`, `h5file,H5ParcellatedArray-mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

`clusters` *Get the cluster map object*

Description

Get the cluster map object

Get the clusters object via generic

Usage

```
clusters(x, ...)

## S4 method for signature 'H5ParcellatedArray'
clusters(x)

## S4 method for signature 'H5ParcellatedMultiScan'
clusters(x)
```

Arguments

`x`	H5ParcellatedMultiScan object
`...`	Additional arguments

Value

The clusters object (e.g., a ClusteredNeuroVol)

The clusters object from the first run.

See Also

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`, `close()`, `cluster_metadata,H5ParcellatedMultiSc`, `h5file,H5ParcellatedArray-method`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

Examples

```
## Not run:
# For H5ParcellatedMultiScan:
if (!is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {
  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(
      master_mask_dims = c(4L,4L,3L),
      num_master_clusters = 2L
```

```
    )
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

    # Get the master cluster map from the experiment
    cluster_vol <- clusters(exp_obj)
    print(cluster_vol)
   # if (requireNamespace("neuroim2", quietly=TRUE)) print(is(cluster_vol, "ClusteredNeuroVol"))

   # Individual runs also have cluster information, potentially accessible via their own methods
    # run1 <- runs(exp_obj)[["Run1_Full"]]
    # run1_clusters <- clusters(run1) # Assuming a method for H5ParcellatedScan
    # print(run1_clusters)

  }, error = function(e) {
    message("clusters example for H5ParcellatedMultiScan failed: ", e$message)
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping clusters example for H5ParcellatedMultiScan: helper not available.")
}

## End(Not run)
```

`configure_memory_warnings` *Configure Memory Usage Warnings*

Description

Control memory usage warnings for large dataset operations in fmristore. These warnings help users understand when operations may consume significant memory.

Usage

```
configure_memory_warnings(enable = TRUE, threshold_mb = 100, verbose = FALSE)
```

Arguments

`enable`	Logical. Whether to enable memory usage warnings (default: TRUE).
`threshold_mb`	Numeric. Memory threshold in MB above which warnings are issued (default: 100).
`verbose`	Logical. Whether to print confirmation of setting changes (default: FALSE).

Details

The memory warning system estimates the size of data that will be loaded into memory and issues warnings when this exceeds the specified threshold. This helps users make informed decisions about subsetting large datasets.

Settings are stored as R options and persist for the current R session: - ‘fmristore.warn_memory‘: Enable/disable warnings - ‘fmristore.memory_threshold_mb‘: Warning threshold in MB

Value

Invisibly returns a list with the current settings:

`enabled`	Logical indicating if warnings are enabled
`threshold_mb`	The warning threshold in MB

Examples

```
# Enable warnings for datasets larger than 50MB
configure_memory_warnings(enable = TRUE, threshold_mb = 50)

# Disable memory warnings
configure_memory_warnings(enable = FALSE)

# Check current settings
list(
  enabled = getOption("fmristore.warn_memory", TRUE),
  threshold_mb = getOption("fmristore.memory_threshold_mb", 100)
)
```

`detect_h5_type` *Detect the type of data in an HDF5 file*

Description

Examines the structure and attributes of an HDF5 file to determine what type of fmristore object it contains.

Usage

```
detect_h5_type(h5obj)
```

Arguments

`h5obj`	An H5File object or file path to an HDF5 file

Value

Character string indicating the type, one of: - "H5NeuroVol" - 3D brain volume - "H5NeuroVec" - 4D brain time series - "H5NeuroVecSeq" - Sequence of 4D scans - "H5ParcellatedMultiScan" - Multi-run parcellated experiment - "H5ParcellatedScan" - Single parcellated run (full data) - "H5ParcellatedScanSummary" - Single parcellated run (summary) - "LatentNeuroVec" - Latent representation - "LabeledVolumeSet" - Labeled brain regions - "unknown" - Type could not be determined

Examples

```
## Not run:
# Detect type from file path
type <- detect_h5_type("data.h5")

# Detect type from open H5File
h5 <- H5File$new("data.h5", "r")
type <- detect_h5_type(h5)
h5$close_all()

## End(Not run)
```

---

get_memory_settings *Get Current Memory Warning Settings*

Description

Retrieve the current memory warning configuration.

Usage

```
get_memory_settings()
```

Value

A list with elements ‘enabled‘ and ‘threshold_mb‘.

Examples

```
# Get current settings
get_memory_settings()
```

H5_ATTRS *HDF5 Attribute Constants*

Description

Named constants for HDF5 attribute names used throughout the package.

Usage

```
H5_ATTRS
```

Format

A named list of character strings.

Value

A named list with HDF5 attribute name constants.

Examples

```
H5_ATTRS$N_TIME
```

H5_DSETS *HDF5 Dataset Name Constants*

Description

Named constants for HDF5 dataset names used throughout the package.

Usage

```
H5_DSETS
```

Format

A named list of character strings.

Value

A named list with HDF5 dataset name constants.

Examples

```
H5_DSETS$DATA
```

H5_PATHS *HDF5 Path Constants*

Description

Central definition of HDF5 file layout constants used throughout the package. This provides a single source of truth for the HDF5 file structure.

Usage

```
H5_PATHS
```

Format

A named list of character strings containing HDF5 path templates.

Value

A named list with HDF5 path constants.

Examples

```
# Access constants
H5_PATHS$SCANS_GROUP # "/scans"
sprintf(H5_PATHS$CLUSTER_DSET_TPL, "run1", 5) # "/scans/run1/clusters/cluster_5"
```

h5file *Get the HDF5 file object*

Description

This generic returns the HDF5 file object associated with the object.

Usage

```
h5file(x)
```

Arguments

x	The object from which to retrieve the HDF5 file object

Value

The HDF5 file object (from package hdf5r)

Examples

```
## Not run:
if (!is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {
  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan()
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

    # Get the H5File object
    h5f <- h5file(exp_obj)
    print(h5f)
    # if (requireNamespace("hdf5r", quietly = TRUE)) print(h5f$is_valid)

  }, error = function(e) {
    message("h5file example failed: ", e$message)
  }, finally = {
    # Closing exp_obj will close the h5file handle it owns
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping h5file example: helper not available.")
}

## End(Not run)
```

---

h5file,H5ParcellatedArray-method

Get the HDF5 file object via generic

Description

Get the HDF5 file object via generic

Get the H5File object for H5ParcellatedScan objects

Get the H5File object for H5ParcellatedScanSummary objects

Usage

```
## S4 method for signature 'H5ParcellatedArray'
h5file(x)

## S4 method for signature 'H5ParcellatedMultiScan'
h5file(x)
```

```
## S4 method for signature 'H5ParcellatedScan'
h5file(x)

## S4 method for signature 'H5ParcellatedScanSummary'
h5file(x)

## S4 method for signature 'H5NeuroVecSeq'
h5file(x)
```

Arguments

`x`	H5ParcellatedScanSummary object

Value

The HDF5 file object from the first run.

The H5File object handle

The H5File object handle

See Also

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`, `close()`, `cluster_metadata,H5ParcellatedMultiSc` `clusters()`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`, `close()`, `cluster_metadata,H5ParcellatedMultiSc` `clusters()`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`, `close()`, `cluster_metadata,H5ParcellatedMultiSc` `clusters()`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

`H5NeuroVec` *H5NeuroVec Constructor*

Description

Constructs an `H5NeuroVec` object, which represents a 4D brain image stored in an HDF5 file. The HDF5 file is opened in read-only mode.

Usage

```
H5NeuroVec(file_name, dataset_name = "data")
```

Arguments

file_name	The path to an existing 4D HDF5 neuroimaging file.
dataset_name	Name of the dataset within the HDF5 file containing the 4D data. Defaults to "data".

Details

This constructor is used for reading existing HDF5 files that conform to the H5NeuroVec specification (typically created via `as_h5("NeuroVec", ...)`).

Value

A new `H5NeuroVec-class` instance with an open HDF5 file handle.

Lifecycle Management

When an `H5NeuroVec` object is created by providing a `file_name`, it opens the specified HDF5 file and maintains an open handle to it. **It is the user's responsibility to explicitly close this handle** when the object is no longer needed to release system resources. This can be done by calling `close(your_h5neurovec_object)`.

Failure to close the handle may lead to issues such as reaching file handle limits or problems with subsequent access to the file.

See Also

`close` for closing the file handle, `NeuroVec-class`

Examples

```
## Not run:
# Assuming "my_vec.h5" is a valid H5NeuroVec HDF5 file
h5vec <- H5NeuroVec("my_vec.h5")
# ... perform operations with h5vec ...
print(dim(h5vec))
# Important: Close the handle when done
close(h5vec)

## End(Not run)
```

H5NeuroVec-class *H5NeuroVec Class*

Description

A class representing a four-dimensional brain image backed by an HDF5 file. H5NeuroVec objects provide efficient storage and access for large-scale 4D neuroimaging data using the HDF5 file format.

Details

H5NeuroVec inherits a space slot from NeuroVec-class, defining the 4D dimensions (e.g., x, y, z, time). Data are stored in an HDF5 dataset and accessed on demand.

Slots

obj An instance of class H5File from the **hdf5r** package, representing the underlying HDF5 file containing the 4D brain image data.

dataset_name A character string specifying the path to the 4D dataset within the HDF5 file. Defaults to "data".

Inheritance

H5NeuroVec inherits from:

- NeuroVec-class: Base class for 4D brain images
- ArrayLike4D-class: Interface for 4D array-like operations

See Also

NeuroVec-class for the base 4D brain image class. H5NeuroVol-class for the 3D counterpart. H5File for details on HDF5 file objects.

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
  requireNamespace("hdf5r", quietly = TRUE) &&
  exists("H5NeuroVec", where = "package:fmristore") &&
  !is.null(fmristore:::create_minimal_h5_for_H5NeuroVec)) {
  # Setup: Create a temporary HDF5 file using a helper
  temp_h5_path <- NULL
  h5_vec <- NULL
  tryCatch({
   temp_h5_path <- fmristore:::create_minimal_h5_for_H5NeuroVec(dims = c(5L, 5L, 3L, 4L))

    # Create an H5NeuroVec object using the constructor
    # Constructor defaults: dataset_name = "data/elements", space from HDF5 /space group
    h5_vec <- fmristore::H5NeuroVec(file_name = temp_h5_path)
```

```
    print(h5_vec)

    # Access a subset of the data
    subset_data <- h5_vec[1:2, 1:2, 1, 1:2]
    print(subset_data)
  }, error = function(e) {
    message("H5NeuroVec example failed: ", e$message)
  }, finally = {
    # Close HDF5 handle owned by H5NeuroVec object
    if (!is.null(h5_vec) && inherits(h5_vec, "H5NeuroVec")) {
      try(close(h5_vec), silent = TRUE)
    }
    # Cleanup temporary file
    if (!is.null(temp_h5_path) && file.exists(temp_h5_path)) {
      unlink(temp_h5_path)
    }
  })
} else {
 message("Skipping H5NeuroVec example: fmristore, neuroim2, hdf5r, or helper not available.")
}

## End(Not run)
```

`H5NeuroVecSeq` *H5NeuroVecSeq constructor*

Description

Load a NeuroVecSeq stored in HDF5 format. Each scan is stored under '/scans/<name>/data' and will be returned as an 'H5NeuroVec'.

Usage

```
H5NeuroVecSeq(file)
```

Arguments

`file`	Path to the HDF5 file created by 'neurovecseq_to_h5()'.

Value

An object of class 'H5NeuroVecSeq'.

Examples

```
## Not run:
# Load a sequence of scans from HDF5
seq_obj <- H5NeuroVecSeq("multi_scan.h5")
n_scans(seq_obj)
scan_names(seq_obj)
close(seq_obj)

## End(Not run)
```

H5NeuroVecSeq-class *H5NeuroVecSeq Class*

Description

Represents a collection of 4D scans stored in a single HDF5 file. Each scan is accessed lazily via an internal H5NeuroVec.

Slots

`obj` An `H5File` handle to the underlying HDF5 file.

`vecs` A list of `H5NeuroVec` objects, one per scan.

H5NeuroVol *H5NeuroVol Constructor*

Description

Constructs an `H5NeuroVol` object representing a 3D brain volume stored in an HDF5 file. The HDF5 file is opened in read-only mode.

Usage

```
H5NeuroVol(file_name)
```

Arguments

`file_name`	A `character` string giving the path to an existing 3D HDF5 neuroimaging file.

Details

This constructor is typically used for reading existing HDF5 files that conform to the H5NeuroVol specification.

Value

A new `H5NeuroVol-class` instance with an open HDF5 file handle.

Lifecycle Management

When an H5NeuroVol object is created by providing a file_name, it opens the specified HDF5 file and maintains an open handle to it. **It is the user's responsibility to explicitly close this handle** when the object is no longer needed to release system resources. This can be done by calling close(your_h5neurovol_object).

As a safety net, a finalizer is registered on the internal HDF5 handle so it will be closed if the object is garbage collected. Explicitly calling close() is still recommended.

Failure to close the handle may lead to issues such as reaching file handle limits or problems with subsequent access to the file.

See Also

close for closing the file handle, NeuroVol-class

Examples

```
## Not run:
# Assuming "my_volume.h5" is a valid H5NeuroVol HDF5 file
h5vol <- H5NeuroVol("my_volume.h5")
# ... perform operations with h5vol ...
print(dim(h5vol))
# Important: Close the handle when done
close(h5vol)

## End(Not run)
```

H5NeuroVol-class *H5NeuroVol Class*

Description

A class representing a three-dimensional brain image backed by an HDF5 dataset. H5NeuroVol objects provide efficient storage and access for large-scale brain imaging data in the HDF5 format.

Details

The H5NeuroVol class inherits a space slot from NeuroVol-class, which specifies the spatial domain (dimensions, orientation, etc.). Data I/O is performed by reading/writing subsets of the HDF5 dataset, allowing efficient handling of large 3D volumes.

Slots

h5obj An instance of class H5File from the **hdf5r** package, representing the underlying HDF5 file containing the brain image data.

Inheritance

H5NeuroVol inherits from:

- NeuroVol-class: Base class for 3D brain volumes
- ArrayLike3D-class: Interface for 3D array-like operations

See Also

NeuroVol-class for the base 3D brain volume class. H5File for details on HDF5 file objects.

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
  requireNamespace("hdf5r", quietly = TRUE) &&
  exists("H5NeuroVol", where = "package:fmristore") && # Check if constructor is available
  !is.null(fmristore:::create_minimal_h5_for_H5NeuroVol)) { # Check helper

  # Setup: Create a temporary HDF5 file using a helper
  # The helper creates a dataset named "data/elements" by default.
  temp_h5_path <- NULL
  h5_vol <- NULL
  tryCatch({
    temp_h5_path <- fmristore:::create_minimal_h5_for_H5NeuroVol(dims = c(5L, 5L, 3L))

    # Create an H5NeuroVol object using the constructor
    # The constructor defaults to dataset_name = "data/elements" if not specified
    # and if a NeuroSpace is not given, it reads it from /space in the HDF5 file.
    h5_vol <- fmristore::H5NeuroVol(file_name = temp_h5_path)

    print(h5_vol)

    # Access a subset of the data
    subset_data <- h5_vol[1:2, 1:2, 1]
    print(subset_data)
  }, error = function(e) {
    message("H5NeuroVol example failed: ", e$message)
  }, finally = {
    # Close HDF5 handle owned by H5NeuroVol object
    if (!is.null(h5_vol) && inherits(h5_vol, "H5NeuroVol")) {
      try(close(h5_vol), silent = TRUE)
    }
    # Cleanup temporary file
    if (!is.null(temp_h5_path) && file.exists(temp_h5_path)) {
      unlink(temp_h5_path)
    }
  })
} else {
  message("Skipping H5NeuroVol example: fmristore, neuroim2, hdf5r, or helper not available.")
}

## End(Not run)
```

H5ParcellatedMultiScan *Constructor for H5ParcellatedMultiScan Objects*

Description

Creates a new 'H5ParcellatedMultiScan' object, representing a collection of clustered neuroimaging runs sharing a common HDF5 file, mask, and cluster map.

This function handles opening the HDF5 file (if a path is provided), identifying available scans, and creating the appropriate run objects ('H5ParcellatedScan' or 'H5ParcellatedScanSummary') for each scan based on the available data within the HDF5 file structure (following the ClusteredTimeSeriesSpec).

Usage

```
H5ParcellatedMultiScan(
  file,
  scan_names = NULL,
  mask = NULL,
  clusters = NULL,
  scan_metadata = NULL,
  cluster_metadata = NULL,
  summary_preference = NULL,
  keep_handle_open = TRUE
)
```

Arguments

file	Either a character string path to the HDF5 file or an open H5File object.
scan_names	(Optional) A character vector specifying which scans under '/scans/' to include in the experiment. If 'NULL' (default), the constructor attempts to discover all available scan groups under '/scans/'.
mask	(Optional) A 'LogicalNeuroVol' object for the brain mask. If 'NULL', the constructor attempts to load it from '/mask' in the HDF5 file.
clusters	(Optional) A 'ClusteredNeuroVol' object for cluster assignments. If 'NULL', the constructor attempts to load it from '/cluster_map' and potentially '/voxel_coords' in the HDF5 file.
scan_metadata	(Optional) A list to override or supplement metadata read from the HDF5 file. If provided, its length should match the number of scans.
cluster_metadata	(Optional) A data.frame to override or supplement cluster metadata read from '/clusters/cluster_meta' in the HDF5 file.

summary_preference
: (Optional) Character string controlling which run type to load. If NULL (default), the constructor reads the /scans@summary_only attribute to choose a default: "require" if summary_only=TRUE, "ignore" if summary_only=FALSE, otherwise "prefer". Explicit values can be "require" (only load summary runs, error if missing), "prefer" (load summary if available, else full), or "ignore" (load full runs only). This influences whether make_run_summary or make_run_full is called. *Note: This parameter requires careful implementation based on HDF5 content checks.*

keep_handle_open
: (Logical) Only relevant if file_source is a path. If TRUE (default), the HDF5 file handle is kept open within the returned object. If FALSE, the handle is closed after reading metadata. *Note:* For most operations, the handle needs to remain open.

Value

A new H5ParcellatedMultiScan object.

See Also

Other H5Parcellated: $,H5ParcellatedMultiScan-method, H5ParcellatedArray-class, H5ParcellatedScan-class, H5ParcellatedScanSummary-class, close(), cluster_metadata,H5ParcellatedMultiScan-method, clusters(), h5file,H5ParcellatedArray-method, mask,H5ParcellatedArray-method, matrix_concat(), n_scans,H5ParcellatedMultiScan-method, scan_metadata,H5ParcellatedMultiScan-method, scan_names,H5ParcellatedMultiScan-method, series_concat()

Examples

```
## Not run:
# Create temporary HDF5 file with minimal experiment structure
temp_file <- tempfile(fileext = ".h5")
exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(file_path = temp_file)

# Create H5ParcellatedMultiScan object
experiment <- H5ParcellatedMultiScan(exp_file)

# Access scan names
print(scan_names(experiment))

# Get number of scans
print(n_scans(experiment))

# Access runs
print(names(experiment@runs))

# Extract data for specific voxels (first 5 mask indices)
voxel_data <- series_concat(experiment, mask_idx = 1:5)
print(dim(voxel_data))

# Clean up
```

```
close(experiment)
unlink(temp_file)

## End(Not run)
```

`H5ParcellatedMultiScan-class` *H5ParcellatedMultiScan Class*

Description

Represents a collection of clustered neuroimaging runs (scans) stored within a single HDF5 file. It acts as a container for 'H5ParcellatedScan' and/or 'H5ParcellatedScanSummary' objects, along with associated metadata.

This class facilitates managing multiple runs that share the same mask and cluster definitions.

Slots

`runs` A 'list' where each element is an object inheriting from 'H5ParcellatedArray' (typically 'H5ParcellatedScan' or 'H5ParcellatedScanSummary').

`scan_metadata` A 'list' containing metadata for each scan in the 'runs' list.

`cluster_metadata` A 'data.frame' containing metadata associated with the clusters.

See Also

`H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`

Other H5Cluster: `[,H5ParcellatedScan,ANY,ANY,ANY-method`, `[,H5ParcellatedScan,ANY,missing,ANY-method`, `as.data.frame()`, `as.matrix()`, `dim()`, `linear_access-methods`, `make_run_full()`, `make_run_summary()`, `series()`, `show,H5ParcellatedScan-method`

`H5ParcellatedScan` *Constructor for H5ParcellatedScan Objects*

Description

Creates a new `H5ParcellatedScan` object, representing a single run of full voxel-level clustered data from an HDF5 file.

This function handles file opening/closing and reads necessary metadata like `n_time` from the HDF5 file if not provided explicitly.

Usage

```
H5ParcellatedScan(
  file,
  scan_name,
  mask,
  clusters,
  n_time = NULL,
  compress = FALSE
)
```

Arguments

file	Character string path to the HDF5 file.
scan_name	The character string name of the scan (e.g., "run1").
mask	A LogicalNeuroVol object for the brain mask.
clusters	A ClusteredNeuroVol object for cluster assignments.
n_time	(Optional) The number of time points. If NULL (default), the function will attempt to read it from the HDF5 file attributes or metadata dataset.
compress	(Optional) Logical indicating compression status (metadata).

Details

Diagnostic messages (such as when n_time is inferred from a dataset) are printed only when the option fmristore.verbose is set to TRUE via options(fmristore.verbose = TRUE).

Value

A new H5ParcellatedScan object with an open file handle managed by the object.

Examples

```
## Not run:
# Create temporary HDF5 file with minimal experiment structure
temp_file <- tempfile(fileext = ".h5")
exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(file_path = temp_file)

# Create mask and clusters
mask <- fmristore:::create_minimal_LogicalNeuroVol(dims = c(5, 5, 4))
clusters <- fmristore:::create_minimal_ClusteredNeuroVol(mask_vol = mask, num_clusters = 3)

# Create H5ParcellatedScan object
run <- H5ParcellatedScan(exp_file, scan_name = "Run1_Full", mask = mask, clusters = clusters)

# Access properties
print(run@n_time)
print(dim(mask))

# Clean up
close(run)
```

```
unlink(temp_file)

## End(Not run)
```

`H5ParcellatedScan-class`

H5ParcellatedScan Class

Description

Represents a single "run" or "scan" of full voxel-level clustered time-series data stored in an HDF5 file. It inherits common properties (mask, clusters, file handle) from 'H5ParcellatedArray' and adds run-specific information.

Slots

`scan_name` A `character` string identifying the scan (e.g., corresponds to a group under '/scans/').

`n_time` An `integer` specifying the number of time points in this run.

`compress` A `logical` indicating if compression was intended or used (metadata). Inherits slots 'obj', 'mask', 'clusters', 'n_voxels' from 'H5ParcellatedArray'.

See Also

`H5ParcellatedArray-class`, `H5ParcellatedScanSummary-class`

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScanSummary-class`, `close()`, `cluster_metadata,H5ParcellatedMultiScan-method`, `clusters()`, `h5file,H5ParcellatedArray-method`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

`H5ParcellatedScanSummary`

Constructor for H5ParcellatedScanSummary Objects

Description

Creates a new `H5ParcellatedScanSummary` object, representing a single run of summary cluster time-series data from an HDF5 file.

This function handles file opening/closing, validates the summary dataset, determines `n_time` from the dataset dimensions, and reconciles cluster names/IDs.

Usage

```
H5ParcellatedScanSummary(
  file,
  scan_name,
  mask,
  clusters = NULL,
  cluster_names = character(),
  cluster_ids = integer(),
  summary_dset = "summary_data"
)
```

Arguments

file	Character string path to the HDF5 file.
scan_name	The character string name of the scan (e.g., "run1").
mask	A LogicalNeuroVol object for the brain mask (used for reference/consistency).
clusters	(Optional) A ClusteredNeuroVol object for cluster assignments. If NULL, cluster names/IDs must be provided or derivable from the dataset.
cluster_names	(Optional) Character vector of names for the clusters.
cluster_ids	(Optional) Integer vector of IDs for the clusters.
summary_dset	(Optional) The name of the dataset within the run's summary group (default: "summary_data").

Value

A new H5ParcellatedScanSummary object with an open file handle managed by the object.

Examples

```
## Not run:
# Create temporary HDF5 file with minimal experiment structure
temp_file <- tempfile(fileext = ".h5")
exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(file_path = temp_file)

# Create mask
mask <- fmristore:::create_minimal_LogicalNeuroVol(dims = c(5, 5, 4))

# Create H5ParcellatedScanSummary object
run_summary <- H5ParcellatedScanSummary(exp_file, scan_name = "Run2_Summary", mask = mask)

# Access properties
print(run_summary@cluster_names)
print(run_summary@n_time)

# Clean up
close(run_summary)
unlink(temp_file)
```

```
## End(Not run)
```

`H5ParcellatedScanSummary-class`
H5ParcellatedScanSummary Class

Description

Represents a single "run" or "scan" containing only *summary* time-series data for clusters (e.g., mean signal per cluster) stored in an HDF5 file. It inherits common properties from ‘H5ParcellatedArray‘ but provides methods suited for accessing summary data (like ‘as.matrix‘) rather than full voxel data.

Slots

`scan_name` A `character` string identifying the scan.

`n_time` An `integer` specifying the number of time points in this run.

`cluster_names` A `character` vector providing names for the clusters (columns in summary matrix).

`cluster_ids` An `integer` vector of cluster IDs corresponding to ‘cluster_names‘.

`summary_dset` A `character` string giving the name of the summary dataset within the run’s HDF5 group (e.g., "summary_data"). Inherits slots ‘obj‘, ‘mask‘, ‘n_voxels‘ from ‘H5ParcellatedArray‘. Note: The ‘clusters‘ slot inherited from ‘H5ParcellatedArray‘ might be NULL or contain the map for reference, but voxel-level access methods are typically disabled/error.

See Also

`H5ParcellatedArray-class`, `H5ParcellatedScan-class`

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `close()`, `cluster_metadata,H5ParcellatedMultiScan-method`, `clusters()`, `h5file,H5ParcellatedArray-method`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `n_scans,H5ParcellatedMultiScan-method`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

`LabeledVolumeSet-class` *LabeledVolumeSet Class*

Description

A class representing a multi-volume dataset stored in HDF5, where each volume is labeled (similar to a named 4th dimension). We store:

- a 3D mask
- a set of label strings
- for each label, data only at the mask's nonzero entries

This extends `NeuroVec`, so it is logically a 4D object with dimension `[X, Y, Z, #labels]`.

Slots

`obj` An `H5File` reference (the file handle).

`mask` A `LogicalNeuroVol` of shape [X, Y, Z].

`labels` A `character` vector for the volume labels.

`load_env` An `environment` storing references for lazy loading.

See Also

`NeuroVec-class`

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
  requireNamespace("hdf5r", quietly = TRUE) &&
  exists("read_labeled_vec", where = "package:fmristore") &&
  !is.null(fmristore:::create_minimal_h5_for_LabeledVolumeSet)) {
  # Setup: Create a temporary HDF5 file suitable for read_labeled_vec
  temp_h5_path <- NULL
  lvs <- NULL
  tryCatch({
    temp_h5_path <- fmristore:::create_minimal_h5_for_LabeledVolumeSet(
      vol_dims = c(5L, 4L, 3L),
      labels = c("CondA", "CondB"),
      num_vols_per_label = 2L
    )

    # Create a LabeledVolumeSet object using the constructor
    lvs <- fmristore::read_labeled_vec(file_name = temp_h5_path)

    print(lvs)
    print(labels(lvs)) # Access labels
```

```
    # Access data for a specific label (returns a NeuroVol or similar)
    # data_cond_a <- lvs[["CondA"]]
    # print(dim(data_cond_a)) # Should be 3D x num_vols_per_label
  }, error = function(e) {
    message("LabeledVolumeSet example failed: ", e$message)
  }, finally = {
    # Close HDF5 handle owned by LabeledVolumeSet object
    if (!is.null(lvs) && inherits(lvs, "LabeledVolumeSet")) {
      try(close(lvs), silent = TRUE)
    }
    # Cleanup temporary file
    if (!is.null(temp_h5_path) && file.exists(temp_h5_path)) {
      unlink(temp_h5_path)
    }
  })
} else {
 message("Skipping LabeledVolumeSet example: fmristore, neuroim2, hdf5r, or helper not available.")
}

## End(Not run)
```

LatentNeuroVec *LatentNeuroVec constructor (re-exported from fmrilatent)*

Description

Creates a latent representation of neuroimaging data using basis functions and loadings. This function is re-exported from the ‘fmrilatent‘ package for convenience.

Usage

```
LatentNeuroVec(
  basis,
  loadings,
  space,
  mask,
  offset = NULL,
  label = "",
  meta = list()
)
```

Arguments

basis	A matrix of temporal embeddings (time x components).
loadings	A matrix of spatial loadings (voxels x components).
space	A ‘NeuroSpace‘ object defining the 4D space.

mask	A 'LogicalNeuroVol' specifying the brain mask.
offset	Optional numeric vector of voxel offsets.
label	Optional character label for the dataset.
meta	Optional named list of additional metadata.

Value

A 'LatentNeuroVec' object.

See Also

LatentNeuroVec in package **fmrilatent** for full documentation.

LatentNeuroVec-io *HDF5 I/O for LatentNeuroVec*

Description

This file provides HDF5 reading and writing functionality for LatentNeuroVec objects.

The class definition and core methods are in the fmrilatent package. This file provides:

- write_vec method for writing to HDF5
- load_data method for reading from HDF5
- validate_latent_file for file validation
- Internal HDF5 helper functions

See Also

LatentNeuroVec (from **fmrilatent**)

---

loadings *Get the loadings matrix (spatial components)*

Description

Get the loadings matrix (spatial components)

Usage

```
loadings(x, ...)
```

Arguments

x	An object, likely a LatentNeuroVec or similar
...	Additional arguments

Value

The loadings matrix (typically voxels x components)

Examples

```
## Not run:
# For LatentNeuroVec:
if (!is.null(fmristore:::create_minimal_LatentNeuroVec)) {
  lnv <- NULL
  tryCatch({
    # Helper creates a mask, n_mask_voxels determined internally or by arg
    lnv <- fmristore:::create_minimal_LatentNeuroVec(
      space_dims = c(4L, 4L, 2L),
      n_time = 10L,
      n_comp = 3L
    )
    l_matrix <- loadings(lnv)
    # Dimensions should be n_voxels_in_mask x n_comp
    print(dim(l_matrix))
  }, error = function(e) {
    message("loadings example for LatentNeuroVec failed: ", e$message)
  })
} else {
  message("Skipping loadings example for LatentNeuroVec: helper not available.")
}

## End(Not run)
```

---

make_run_full *Constructor for H5ParcellatedScan Objects*

Description

Creates a new ‘H5ParcellatedScan‘ object, representing a single run of full voxel-level clustered data from an HDF5 file.

It performs necessary validation and can read metadata like ‘n_time‘ directly from the HDF5 file attributes or metadata dataset.

Usage

```
make_run_full(
  file_source,
  scan_name,
  mask,
  clusters,
  n_time = NULL,
  compress = FALSE
)
```

Arguments

file_source	Either an open 'H5File' object or a character string path to the HDF5 file.
scan_name	The character string name of the scan (e.g., "run1").
mask	A 'LogicalNeuroVol' object for the brain mask.
clusters	A 'ClusteredNeuroVol' object for cluster assignments.
n_time	(Optional) The number of time points. If 'NULL' (default), the function will attempt to read it from the HDF5 file attributes (e.g., '/scans/<scan_name>@n_time' or '/scans/<scan_name>/metadata/n_time').
compress	(Optional) Logical indicating compression status (metadata).

Value

A new 'H5ParcellatedScan' object.

See Also

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_summary(), series(), show,H5ParcellatedScan-method

Examples

```
## Not run:
# Note: This function is deprecated in favor of H5ParcellatedScan()

# Create temporary HDF5 file
temp_file <- tempfile(fileext = ".h5")
exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(file_path = temp_file)

# Create mask and clusters
mask <- fmristore:::create_minimal_LogicalNeuroVol(dims = c(5, 5, 4))
clusters <- fmristore:::create_minimal_ClusteredNeuroVol(mask_vol = mask, num_clusters = 3)

# Create run object (deprecated - will show deprecation warning)
suppressWarnings({
  run <- make_run_full(exp_file, scan_name = "Run1_Full", mask = mask, clusters = clusters)
})

# Clean up - note: make_run_full returns an object that doesn't own the file handle
unlink(temp_file)

## End(Not run)
```

make_run_summary *Constructor for H5ParcellatedScanSummary Objects*

Description

Creates a new 'H5ParcellatedScanSummary' object, representing a single run of summary cluster time-series data from an HDF5 file.

It performs validation, checks for the existence of the summary dataset, and determines 'n_time' from the dataset dimensions.

Usage

```
make_run_summary(
  file_source,
  scan_name,
  mask,
  clusters,
  cluster_names = character(),
  cluster_ids = integer(),
  summary_dset = "summary_data"
)
```

Arguments

file_source	Either an open 'H5File' object or a character string path to the HDF5 file.
scan_name	The character string name of the scan (e.g., "run1").
mask	A 'LogicalNeuroVol' object for the brain mask (used for reference/consistency).
clusters	A 'ClusteredNeuroVol' object for cluster assignments. This is required for consistency and reference, even though summary data is accessed.
cluster_names	(Optional) Character vector of names for the clusters.
cluster_ids	(Optional) Integer vector of IDs for the clusters.
summary_dset	(Optional) The name of the dataset within the run's summary group (default: "summary_data").

Value

A new 'H5ParcellatedScanSummary' object.

See Also

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), series(), show,H5ParcellatedScan-method

Examples

```
## Not run:
# Note: This function is deprecated in favor of H5ParcellatedScanSummary()

# Create temporary HDF5 file
temp_file <- tempfile(fileext = ".h5")
exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(file_path = temp_file)

# Create mask
mask <- fmristore:::create_minimal_LogicalNeuroVol(dims = c(5, 5, 4))

# Create summary run object (deprecated - will show deprecation warning)
suppressWarnings({
  run_summary <- make_run_summary(exp_file, scan_name = "Run2_Summary", mask = mask)
})

# Clean up - note: make_run_summary returns an object that doesn't own the file handle
unlink(temp_file)

## End(Not run)
```

---

map *Get the index map volume*

Description

Get the index map volume

Usage

```
map(x, ...)
```

Arguments

x	An object with an index map, like LatentNeuroVec
...	Additional arguments

Value

The index map object (e.g., an IndexLookupVol from neuroim2)

Examples

```
## Not run:
# For LatentNeuroVec:
if (!is.null(fmristore:::create_minimal_LatentNeuroVec)) {
  lnv <- NULL
  tryCatch({
```

```
    lnv <- fmristore:::create_minimal_LatentNeuroVec(space_dims = c(3L,3L,2L))
    map_vol <- map(lnv)
    print(map_vol)
   # if (requireNamespace("neuroim2", quietly=TRUE)) print(is(map_vol, "IndexLookupVol"))
  }, error = function(e) {
    message("map example for LatentNeuroVec failed: ", e$message)
  })
} else {
  message("Skipping map example for LatentNeuroVec: helper not available.")
}

## End(Not run)
```

---

mask *Get the mask volume*

Description

Get the mask volume

Arguments

x	An object with a mask, like LatentNeuroVec or H5ParcellatedMultiScan
...	Additional arguments

Value

The mask object (e.g., a LogicalNeuroVol)

Examples

```
## Not run:
# For LatentNeuroVec:
if (!is.null(fmristore:::create_minimal_LatentNeuroVec)) {
  lnv <- NULL
  tryCatch({
    lnv <- fmristore:::create_minimal_LatentNeuroVec(space_dims = c(3L,3L,2L))
    mask_vol <- mask(lnv)
    print(mask_vol)
   # if (requireNamespace("neuroim2", quietly=TRUE)) print(is(mask_vol, "LogicalNeuroVol"))
  }, error = function(e) {
    message("mask example for LatentNeuroVec failed: ", e$message)
  })
} else {
  message("Skipping mask example for LatentNeuroVec: helper not available.")
}

# For H5ParcellatedMultiScan:
if (!is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {
```

```
  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan()
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)
    mask_vol_exp <- mask(exp_obj)
    print(mask_vol_exp)
  }, error = function(e) {
    message("mask example for H5ParcellatedMultiScan failed: ", e$message)
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping mask example for H5ParcellatedMultiScan: helper not available.")
}

## End(Not run)
```

`mask,H5ParcellatedArray-method`

Get the mask object via generic

Description

Get the mask object via generic

Usage

```
## S4 method for signature 'H5ParcellatedArray'
mask(x)

## S4 method for signature 'H5ParcellatedMultiScan'
mask(x)

## S4 method for signature 'LatentNeuroVec'
mask(x)
```

Arguments

x	H5ParcellatedMultiScan object

Value

The mask object from the first run.

See Also

Other H5Parcellated: $,H5ParcellatedMultiScan-method, H5ParcellatedArray-class, H5ParcellatedMultiScan, H5ParcellatedScan-class, H5ParcellatedScanSummary-class, close(), cluster_metadata,H5ParcellatedMultiSc clusters(), h5file,H5ParcellatedArray-method, matrix_concat(), n_scans,H5ParcellatedMultiScan-method, scan_metadata,H5ParcellatedMultiScan-method, scan_names,H5ParcellatedMultiScan-method, series_concat()

`matrix_concat` *Concatenate Cluster Summary Matrices Across Runs (Generic)*

Description

Concatenate Cluster Summary Matrices Across Runs (Generic)

Usage

```
matrix_concat(experiment, run_indices = NULL, ...)

## S4 method for signature 'H5ParcellatedMultiScan'
matrix_concat(experiment, run_indices = NULL)
```

Arguments

`experiment`	The experiment object (typically H5ParcellatedMultiScan-class).
`run_indices`	Optional: A numeric or character vector specifying which runs to include. If NULL, uses all runs that are of summary type or can produce a summary matrix.
`...`	Additional arguments for methods.

Value

A concatenated matrix (typically time x clusters).

Methods (by class)

- `matrix_concat(H5ParcellatedMultiScan)`: Concatenate summary matrices for H5ParcellatedMultiScan

See Also

Other H5Parcellated: $,H5ParcellatedMultiScan-method, H5ParcellatedArray-class, H5ParcellatedMultiScan, H5ParcellatedScan-class, H5ParcellatedScanSummary-class, close(), cluster_metadata,H5ParcellatedMultiSc clusters(), h5file,H5ParcellatedArray-method, mask,H5ParcellatedArray-method, n_scans,H5ParcellatedMult scan_metadata,H5ParcellatedMultiScan-method, scan_names,H5ParcellatedMultiScan-method, series_concat()

Examples

```
## Not run:
if (!is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {
  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(
      n_time_run2 = 6L # Shorter time series for Run2_Summary
    )
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

   # Note: matrix_concat typically works on runs with summary data (H5ParcellatedScanSummary)
    # The helper creates "Run2_Summary".
    # concatenated_matrix <- matrix_concat(exp_obj, run_indices = "Run2_Summary")
    # print(dim(concatenated_matrix))
    # Should be n_time_run2 x n_master_clusters (e.g., 6x3 if num_master_clusters is 3)
    # print(head(concatenated_matrix))

    # The method should correctly identify and use the summary run.
    # If only one summary run is present, run_indices can often be omitted.
    conc_matrix <- matrix_concat(exp_obj, run_indices = "Run2_Summary")
    print(paste("Dimensions of concatenated matrix from Run2_Summary:",
                paste(dim(conc_matrix), collapse="x")))

    # Example with all compatible runs (should pick up Run2_Summary)
    # conc_matrix_all <- matrix_concat(exp_obj)
    # print(paste("Dimensions of concatenated matrix from all compatible runs:",
    #             paste(dim(conc_matrix_all), collapse="x")))

  }, error = function(e) {
    message("matrix_concat example failed: ", e$message)
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping matrix_concat example: helper not available.")
}

## End(Not run)
```

---

n_scans *Get the number of scans*

Description

This generic returns the number of scans in an object (e.g. a sequence of fMRI scans).

Usage

```
n_scans(x)
```

Arguments

x	The object from which to retrieve the number of scans

Value

An integer representing the number of scans

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("hdf5r", quietly = TRUE) &&
    exists("H5ParcellatedMultiScan", where = "package:fmristore") &&
    exists("n_scans", where = "package:fmristore") &&
    !is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {

  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    # Create a minimal H5ParcellatedMultiScan which contains runs (scans)
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan()
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

    # Get the number of scans
    num_scans <- n_scans(exp_obj)
    print(num_scans) # Should be 2 based on the helper

  }, error = function(e) {
    message("n_scans example failed: ", e$message)
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping n_scans example: dependencies or helper not available.")
}

## End(Not run)
```

n_scans,H5ParcellatedMultiScan-method *Get number of scans*

Description

Get number of scans

Usage

```
## S4 method for signature 'H5ParcellatedMultiScan'
n_scans(x)

## S4 method for signature 'H5NeuroVecSeq'
n_scans(x)
```

Arguments

x	H5ParcellatedMultiScan object

Value

Integer number of scans.

See Also

Other H5Parcellated: `$,H5ParcellatedMultiScan-method`, `H5ParcellatedArray-class`, `H5ParcellatedMultiScan`, `H5ParcellatedScan-class`, `H5ParcellatedScanSummary-class`, `close()`, `cluster_metadata,H5ParcellatedMultiSc`, `clusters()`, `h5file,H5ParcellatedArray-method`, `mask,H5ParcellatedArray-method`, `matrix_concat()`, `scan_metadata,H5ParcellatedMultiScan-method`, `scan_names,H5ParcellatedMultiScan-method`, `series_concat()`

neurovecseq_to_h5 *Convert NeuroVecSeq to HDF5*

Description

Writes a NeuroVecSeq object (sequence of multiple 4D neuroimaging scans) to an HDF5 file. Each scan is stored separately within the file, allowing for different time dimensions while maintaining the same spatial dimensions across all scans.

Usage

```
neurovecseq_to_h5(
  neurovecseq,
  file = NULL,
  scan_names = NULL,
  data_type = "FLOAT",
  chunk_dim = NULL,
  compression = 6,
  scan_metadata = NULL
)
```

Arguments

neurovecseq	A NeuroVecSeq object containing multiple NeuroVec objects
file	Path to the output HDF5 file. If NULL, uses a temporary file.
scan_names	Optional character vector of names for each scan. If NULL, uses "scan_1", "scan_2", etc.
data_type	Character string specifying the data type: "FLOAT" (default), "DOUBLE", or "INT"
chunk_dim	Numeric vector specifying chunk dimensions for HDF5 storage. If NULL, uses time-optimized chunking (small spatial chunks, full time dimension).
compression	Integer compression level 0-9. Default is 6.
scan_metadata	Optional named list where each element is a list of metadata for the corresponding scan. Names should match scan_names.

Details

The HDF5 file structure created is:

- `/` - Root group with attributes "rtype" = "NeuroVecSeq" and "n_scans"
- `/space/` - Group containing shared spatial information
- `/space/dim` - Spatial dimensions (3D)
- `/space/origin` - Spatial origin
- `/space/trans` - Spatial transformation matrix
- `/scans/` - Group containing all scans
- `/scans/<scan_name>/` - Group for each scan with "n_time" attribute
- `/scans/<scan_name>/data` - 4D data array for the scan
- `/scans/<scan_name>/metadata/` - Optional metadata group

Value

The file path of the created HDF5 file

Examples

```
## Not run:
# Create example NeuroVec objects with different time dimensions
vec1 <- NeuroVec(
  array(rnorm(10 * 10 * 5 * 20), dim = c(10, 10, 5, 20)),
  NeuroSpace(c(10, 10, 5, 20))
)
vec2 <- NeuroVec(
  array(rnorm(10 * 10 * 5 * 30), dim = c(10, 10, 5, 30)),
  NeuroSpace(c(10, 10, 5, 30))
)

# Create NeuroVecSeq
nvs <- NeuroVecSeq(vec1, vec2)

# Convert to HDF5
h5_file <- neurovecseq_to_h5(nvs,
  scan_names = c("rest_run1", "task_run1"),
  scan_metadata = list(
    rest_run1 = list(TR = 2.0, task = "rest"),
    task_run1 = list(TR = 2.0, task = "motor")
  )
)

# File can later be read back (functionality to be implemented)
unlink(h5_file)

## End(Not run)
```

`offset` *Get the offset vector*

Description

Get the offset vector

Arguments

`x`	An object, likely a LatentNeuroVec or similar
`...`	Additional arguments

Value

The offset vector

Examples

```
## Not run:
# For LatentNeuroVec:
if (!is.null(fmristore:::create_minimal_LatentNeuroVec)) {
  lnv <- NULL
  tryCatch({
    lnv <- fmristore:::create_minimal_LatentNeuroVec(
      space_dims = c(3L, 3L, 2L),
      n_mask_voxels = 4L # Specify a small number of voxels in mask
    )
    off_vector <- offset(lnv)
    print(head(off_vector))
    # The LatentNeuroVec constructor (used by helper) defaults to zero offset.
    # Length should be n_voxels_in_mask.
    if (inherits(lnv, "LatentNeuroVec")) {
       # Assuming lnv@mask is a LogicalNeuroVol created by the helper
       # The offset vector length should match the number of TRUE voxels in the mask.
       # print(length(off_vector) == sum(lnv@mask@.Data))
    }
  }, error = function(e) {
    message("offset example for LatentNeuroVec failed: ", e$message)
  })
} else {
  message("Skipping offset example for LatentNeuroVec: helper not available.")
}

## End(Not run)
```

---

read_dataset *Read Dataset from HDF5*

Description

Generic function for reading neuroimaging datasets from HDF5 format

Usage

```
read_dataset(x, ...)

## S4 method for signature 'H5File'
read_dataset(x, type = NULL, ...)
```

Arguments

x	The file path or H5File object to read from
...	Additional arguments passed to methods
type	Optional character string specifying the expected type. If NULL (default), the type is auto-detected.

Value

The appropriate fmristore object

Examples

```
## Not run:
# Auto-detect type from HDF5 file
obj <- read_dataset("data.h5")

# Specify type explicitly
vec <- read_dataset("data.h5", type = "H5NeuroVec")

## End(Not run)
```

`read_dataset,character-method`
Read Dataset from HDF5 File

Description

Reads a dataset from an HDF5 file with automatic type detection. The function examines the file structure to determine the appropriate object type and returns the corresponding fmristore object.

Usage

```
## S4 method for signature 'character'
read_dataset(x, type = NULL, ...)
```

Arguments

`x`	Path to HDF5 file.
`type`	Optional character string specifying the expected type. If NULL (default), the type is auto-detected.
`...`	Additional arguments passed to the object constructor

Value

An fmristore object of the appropriate type: - `H5NeuroVol` for 3D volumes - `H5NeuroVec` for 4D time series - `H5NeuroVecSeq` for sequences of 4D data - `H5ParcellatedMultiScan` for parcellated experiments - `LatentNeuroVec` for latent representations - `LabeledVolumeSet` for labeled regions

Examples

```
## Not run:
# Auto-detect type
obj <- read_dataset("data.h5")

# Specify type explicitly
obj <- read_dataset("data.h5", type = "H5NeuroVec")

# Pass additional arguments to constructor
obj <- read_dataset("data.h5", dataset_name = "processed")

## End(Not run)
```

---

read_labeled_vec *Read a Labeled Neuroimaging Volume Set from HDF5*

Description

Reads an HDF5 file, typically one previously created by `write_labeled_vec` (now deprecated), and constructs a `LabeledVolumeSet-class` object. The HDF5 file is opened in read-only mode.

Usage

```
read_labeled_vec(file_path)
```

Arguments

`file_path`	Character string: path to the HDF5 file.

Details

This function is the primary way to load data into a `LabeledVolumeSet` from its HDF5 disk representation. The structure of the HDF5 file is expected to follow the specification laid out by `write_labeled_vec`.

Value

A `LabeledVolumeSet-class` object with an open HDF5 file handle.

Lifecycle Management

When a `LabeledVolumeSet` object is created by this function, it opens the specified HDF5 file and the returned object takes ownership of this open file handle. **It is the user's responsibility to explicitly close this handle** when the object is no longer needed to release system resources. This can be done by calling `close(your_labeled_volume_set_object)`.

Failure to close the handle may lead to issues such as reaching file handle limits or problems with subsequent access to the file.

See Also

write_labeled_vec for the (deprecated) writing function, LabeledVolumeSet-class for details on the object structure, close for closing the file handle.

Examples

```
## Not run:
# Assuming "my_labeled_set.h5" is a valid HDF5 file for LabeledVolumeSet
lvs <- read_labeled_vec("my_labeled_set.h5")
# ... perform operations with lvs ...
print(dim(lvs))
print(labels(lvs))
# Important: Close the handle when done
close(lvs)

## End(Not run)
```

---

scan_metadata *Get scan metadata*

Description

This generic returns any available metadata associated with each scan.

Usage

```
scan_metadata(x)
```

Arguments

x	The object from which to retrieve the scan metadata

Value

A list (or other structure) containing metadata for each scan

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("hdf5r", quietly = TRUE) &&
    exists("H5ParcellatedMultiScan", where = "package:fmristore") &&
    exists("scan_metadata", where = "package:fmristore") &&
    !is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {

  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    # Create a minimal H5ParcellatedMultiScan
```

```
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan()
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

    # Get the scan metadata
    s_meta <- scan_metadata(exp_obj)
    print(s_meta)
    # The helper currently doesn't add rich scan_metadata,
    # so this might be an empty list or list of NULLs by default.
    # length(s_meta) == n_scans(exp_obj) # This should hold TRUE

  }, error = function(e) {
    message("scan_metadata example failed: ", e$message)
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping scan_metadata example: dependencies or helper not available.")
}

## End(Not run)
```

`scan_metadata,H5ParcellatedMultiScan-method`
Get scan metadata

Description

Get scan metadata

Usage

```
## S4 method for signature 'H5ParcellatedMultiScan'
scan_metadata(x)
```

Arguments

x	H5ParcellatedMultiScan object

Value

List of scan metadata.

See Also

Other H5Parcellated: $,H5ParcellatedMultiScan-method, H5ParcellatedArray-class, H5ParcellatedMultiScan, H5ParcellatedScan-class, H5ParcellatedScanSummary-class, close(), cluster_metadata,H5ParcellatedMultiSc clusters(), h5file,H5ParcellatedArray-method, mask,H5ParcellatedArray-method, matrix_concat(), n_scans,H5ParcellatedMultiScan-method, scan_names,H5ParcellatedMultiScan-method, series_concat()

`scan_names` *Get the scan names*

Description

This generic returns a character vector of scan names or labels.

Usage

```
scan_names(x)
```

Arguments

x	The object from which to retrieve the scan names

Value

A character vector of scan names

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
    requireNamespace("hdf5r", quietly = TRUE) &&
    exists("H5ParcellatedMultiScan", where = "package:fmristore") &&
    exists("scan_names", where = "package:fmristore") &&
    !is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {

  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    # Create a minimal H5ParcellatedMultiScan
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan()
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

    # Get the scan names
    s_names <- scan_names(exp_obj)
    print(s_names) # Should be c("Run1_Full", "Run2_Summary") or similar

  }, error = function(e) {
    message("scan_names example failed: ", e$message)
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
```

```
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping scan_names example: dependencies or helper not available.")
}

## End(Not run)
```

---

scan_names,H5ParcellatedMultiScan-method
Get scan names

Description

Get scan names

Usage

```
## S4 method for signature 'H5ParcellatedMultiScan'
scan_names(x)

## S4 method for signature 'H5NeuroVecSeq'
scan_names(x)
```

Arguments

x	H5ParcellatedMultiScan object

Value

Character vector of scan names.

See Also

Other H5Parcellated: $,H5ParcellatedMultiScan-method, H5ParcellatedArray-class, H5ParcellatedMultiScan, H5ParcellatedScan-class, H5ParcellatedScanSummary-class, close(), cluster_metadata,H5ParcellatedMultiSc clusters(), h5file,H5ParcellatedArray-method, mask,H5ParcellatedArray-method, matrix_concat(), n_scans,H5ParcellatedMultiScan-method, scan_metadata,H5ParcellatedMultiScan-method, series_concat()

series *Extract Time Series Data*

Description

A generic function to extract time series data from an object. Specific methods will determine how data is extracted based on the object's class and the indices provided.

Usage

```
series(x, i, ...)

## S4 method for signature 'H5ParcellatedScan,numeric'
series(x, i, j, k, ...)

## S4 method for signature 'H5ParcellatedScan,matrix'
series(x, i, ...)

## S4 method for signature 'H5ParcellatedScanSummary,ANY'
series(x, i, ...)

## S4 method for signature 'H5NeuroVec,matrix'
series(x, i)

## S4 method for signature 'H5NeuroVec,integer'
series(x, i, j, k, drop = TRUE)

## S4 method for signature 'H5NeuroVec,numeric'
series(x, i, j, k, drop = TRUE)
```

Arguments

x	The object from which to extract series data.
i	An index, typically specifying voxels or elements. The interpretation of 'i' depends on the specific method. It can be numeric indices, a matrix of coordinates, etc.
...	Additional arguments passed to specific methods (e.g., 'drop').
j	Optional y-coordinate or further index specification used by some methods.
k	Optional z-coordinate or further index specification used by some methods.
drop	Whether to drop dimensions of length 1 (default TRUE).

Value

A matrix or vector containing the time series data. The exact format (e.g., time x voxels, or a simple vector for a single voxel) depends on the method and arguments.

See Also

Methods for this generic are available for classes like H5ParcellatedScan.

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), show,H5ParcellatedScan-method

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), show,H5ParcellatedScan-method

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), show,H5ParcellatedScan-method

Examples

```
# This is a generic function; examples are provided with its specific methods.
# For example, see help("series,H5ParcellatedScan-method")
```

---

series_concat *Concatenate Voxel Time Series Across Runs (Generic)*

Description

Concatenate Voxel Time Series Across Runs (Generic)

Usage

```
series_concat(experiment, mask_idx, run_indices = NULL, ...)

## S4 method for signature 'H5ParcellatedMultiScan,numeric'
series_concat(experiment, mask_idx, run_indices = NULL)
```

Arguments

experiment	The experiment object (typically H5ParcellatedMultiScan-class).
mask_idx	Indices of voxels within the mask of the experiment.
run_indices	Optional: A numeric or character vector specifying which runs to include.
...	Additional arguments for methods.

Value

A concatenated matrix (typically time x voxels).

Methods (by class)

- series_concat(experiment = H5ParcellatedMultiScan, mask_idx = numeric): Concatenate voxel time series for H5ParcellatedMultiScan

See Also

Other H5Parcellated: $,H5ParcellatedMultiScan-method, H5ParcellatedArray-class, H5ParcellatedMultiScan, H5ParcellatedScan-class, H5ParcellatedScanSummary-class, close(), cluster_metadata,H5ParcellatedMultiSc clusters(), h5file,H5ParcellatedArray-method, mask,H5ParcellatedArray-method, matrix_concat(), n_scans,H5ParcellatedMultiScan-method, scan_metadata,H5ParcellatedMultiScan-method, scan_names,H5ParcellatedMultiScan-method

Examples

```
## Not run:
if (!is.null(fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan)) {
  temp_exp_file <- NULL
  exp_obj <- NULL
  tryCatch({
    temp_exp_file <- fmristore:::create_minimal_h5_for_H5ParcellatedMultiScan(
      master_mask_dims = c(4L,4L,2L), # Smaller mask for example
      n_time_run1 = 5L # Shorter time series for Run1_Full
    )
    exp_obj <- fmristore::H5ParcellatedMultiScan(file_path = temp_exp_file)

    # Get some valid voxel indices from the mask
    # The master mask is created by the helper.
    # exp_mask <- mask(exp_obj)
    # valid_mask_indices <- which(exp_mask@.Data)
    # For simplicity, let's assume first few indices if mask is not empty.
    # If mask has at least 2 voxels:
    # selected_vox_indices <- valid_mask_indices[1:min(2, length(valid_mask_indices))]
    # A more robust way for an example without directly loading mask here:
    # The H5ParcellatedMultiScan has a mask, and its linear indices are 1:sum(mask_data)
    # For a 4x4x2 mask, if 50% are true, there are 16 true voxels. Indices are 1 to 16.
    # Let's pick first 2 voxels in the mask's internal indexing (1-based).

    # Note: series_concat typically works on runs with full data (H5ParcellatedScan)
    # The helper creates "Run1_Full".
    # concatenated_series <- series_concat(exp_obj, mask_idx = c(1, 2),
    #                                      run_indices = "Run1_Full")
    # print(dim(concatenated_series)) # Should be n_time_run1 x 2
    # print(head(concatenated_series))

   # For a fully runnable example, need to ensure mask_idx is valid for the created object.
   # Since the helper creates a mask, we can try to use mask_idx = 1 (first voxel in mask).
    # The `series_concat` method for H5ParcellatedMultiScan handles this.
    if (n_voxels(exp_obj) > 0) { # n_voxels from H5ParcellatedArray slot
       conc_series <- series_concat(exp_obj, mask_idx = 1, run_indices = "Run1_Full")
       print(paste("Dimensions of concatenated series for voxel 1 from Run1_Full:",
                   paste(dim(conc_series), collapse="x")))
    } else {
       message("Skipping series_concat demonstration as experiment mask is empty.")
    }

  }, error = function(e) {
    message("series_concat example failed: ", e$message)
```

```
  }, finally = {
    if (!is.null(exp_obj)) try(close(exp_obj), silent = TRUE)
    if (!is.null(temp_exp_file) && file.exists(temp_exp_file)) {
      unlink(temp_exp_file)
    }
  })
} else {
  message("Skipping series_concat example: helper not available.")
}

## End(Not run)
```

---

show,H5NeuroVec-method

Display an H5NeuroVec object

Description

Prints a concise summary of the H5NeuroVec object, including 4D dimensions, spacing, origin, and orientation.

Usage

```
## S4 method for signature 'H5NeuroVec'
show(object)
```

Arguments

object	An H5NeuroVec object to show.

---

show,H5NeuroVol-method

Show Method for H5NeuroVol

Description

Displays a summary of the H5NeuroVol object, including dimensions, spacing, origin, and HDF5 file information, without reading voxel data.

Usage

```
## S4 method for signature 'H5NeuroVol'
show(object)
```

Arguments

object	The H5NeuroVol object to display.

show,H5ParcellatedScan-method

Show Method for H5ParcellatedScanSummary

Description

Show Method for H5ParcellatedScanSummary

Usage

```
## S4 method for signature 'H5ParcellatedScan'
show(object)

## S4 method for signature 'H5ParcellatedScanSummary'
show(object)

## S4 method for signature 'H5ParcellatedMultiScan'
show(object)
```

Arguments

object	An H5ParcellatedScanSummary object

See Also

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), series()

Other H5Cluster: H5ParcellatedMultiScan-class, [,H5ParcellatedScan,ANY,ANY,ANY-method, [,H5ParcellatedScan,ANY,missing,ANY-method, as.data.frame(), as.matrix(), dim(), linear_access-methods, make_run_full(), make_run_summary(), series()

show,LabeledVolumeSet-method

show method for LabeledVolumeSet

Description

Displays essential info about a LabeledVolumeSet, including spatial dims, number of volumes, label previews, spacing, origin, orientation (if known), and storage paths.

Usage

```
## S4 method for signature 'LabeledVolumeSet'
show(object)
```

Arguments

object	A LabeledVolumeSet-class instance

---

validate_latent_file *Validate HDF5 File Against Latent Spec*

Description

Performs basic checks on an HDF5 file to verify if it conforms to the essential structure and dimension consistency defined by the 'BasisEmbeddingSpec.yaml' specification.

Usage

```
validate_latent_file(file_path)
```

Arguments

file_path	Character string specifying the path to the HDF5 file.

Details

Checks performed:

- File existence and HDF5 readability.
- Presence of mandatory groups: '/header', '/basis', '/scans'.
- Presence of mandatory datasets: '/header/dim', '/mask', '/basis/basis_matrix', at least one scan group under '/scans', and its 'embedding' dataset.
- Dimension consistency between header, mask, basis, and embedding.

Value

Logical 'TRUE' if basic checks pass, 'FALSE' otherwise. Issues warnings for inconsistencies found. Throws an error if the file cannot be opened or fundamental groups/datasets are missing.

`write_cluster_result` *Write Clustering Results to HDF5*

Description

A convenience function for writing clustering results (e.g., from supervoxels()) along with the original neuroimaging data to an HDF5 file. This function serves as a user-friendly interface to `write_dataset`.

Usage

```
write_cluster_result(
  cluster_result,
  neurovec,
  filename,
  scan_name = "scan_001",
  type = c("full", "summary"),
  cluster_metadata = NULL,
  overwrite = FALSE,
  compress = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`cluster_result`	Either a `ClusteredNeuroVol` object or a list containing: • `clusters`: Numeric vector of cluster assignments for each voxel within the mask • `mask`: A `LogicalNeuroVol` defining which voxels were included in clustering • `metadata`: (Optional) Additional cluster information
`neurovec`	A `NeuroVec` object containing the 4D fMRI time series data
`filename`	Character string specifying the output HDF5 file path
`scan_name`	Character string naming this scan in the HDF5 file (default: "scan_001")
`type`	Character string, either "full" for voxel-level data or "summary" for cluster-averaged data
`cluster_metadata`	Optional data.frame containing metadata for each cluster. Should have a column named `cluster_id` matching the unique cluster IDs.
`overwrite`	Logical, whether to overwrite an existing file (default: FALSE)
`compress`	Logical or numeric. If logical: TRUE uses compression level 4, FALSE uses no compression. If numeric: specify compression level 0-9.
`verbose`	Logical, whether to print progress messages (default: TRUE)
`...`	Additional arguments passed to `write_dataset`

Details

This function bridges clustering results from various sources (e.g., `supervoxels()`, custom clustering algorithms) with the fmristore HDF5 storage system. It automatically converts the clustering information into the required `ClusteredNeuroVol` format and delegates to `write_dataset()` for efficient HDF5 storage.

When `type = "full"`, the complete voxel-level time series are stored organized by cluster. When `type = "summary"`, only the cluster-averaged time series are stored, resulting in much smaller file sizes.

Value

Invisibly returns NULL. Called for its side effect of creating an HDF5 file.

See Also

`write_dataset` for the underlying implementation `read_dataset` for loading the saved data `ClusteredNeuroVol` for the cluster representation

Examples

```
## Not run:
# Example with clustering results from supervoxels()
clust_result <- supervoxels(brain_data, n_clusters = 100)

# Write full voxel-level data
write_cluster_result(
  cluster_result = clust_result,
  neurovec = my_4d_data,
  filename = "clustered_data.h5",
  type = "full"
)

# Write only cluster averages (more compact)
write_cluster_result(
  cluster_result = clust_result,
  neurovec = my_4d_data,
  filename = "clustered_summary.h5",
  type = "summary",
  compress = 6 # Higher compression
)

# With cluster metadata
cluster_info <- data.frame(
  cluster_id = 1:100,
  region = rep(c("frontal", "parietal", "temporal", "occipital"), 25),
  size = sapply(1:100, function(i) sum(clust_result$clusters == i))
)

write_cluster_result(
  cluster_result = clust_result,
  neurovec = my_4d_data,
```

```
  filename = "clustered_with_metadata.h5",
  cluster_metadata = cluster_info,
  type = "full"
)

## End(Not run)
```

`write_dataset` *Write Dataset to HDF5*

Description

Generic function for writing neuroimaging datasets to HDF5 format

Write a single NeuroVec object with clustering to HDF5

Write multiple NeuroVec objects with clustering to HDF5

Export an existing H5ParcellatedMultiScan object to a new HDF5 file

Export an existing H5NeuroVec object to a new HDF5 file

Export an existing H5NeuroVol object to a new HDF5 file

Write a LatentNeuroVec object to HDF5

Write a neuroim2::ClusteredNeuroVec object to HDF5 using fmristore's clustered format

Usage

```
write_dataset(x, ...)

## S4 method for signature 'NeuroVec'
write_dataset(
  x,
  file,
  clusters,
  mask = NULL,
  summary = FALSE,
  summary_fun = mean,
  scan_name = "scan_001",
  compression = 4,
  ...
)

## S4 method for signature 'list'
write_dataset(
  x,
  file,
  clusters,
  mask = NULL,
```

```
  summary = FALSE,
  summary_fun = mean,
  scan_names = NULL,
  compression = 4,
  ...
)

## S4 method for signature 'H5ParcellatedMultiScan'
write_dataset(x, file, overwrite = FALSE, ...)

## S4 method for signature 'H5NeuroVec'
write_dataset(x, file, overwrite = FALSE, ...)

## S4 method for signature 'H5NeuroVol'
write_dataset(x, file, overwrite = FALSE, ...)

## S4 method for signature 'LatentNeuroVec'
write_dataset(x, file, compression = 6, ...)

## S4 method for signature 'ClusteredNeuroVec'
write_dataset(
  x,
  file,
  scan_name = "scan_001",
  as_multiscan = FALSE,
  compression = 4,
  ...
)
```

Arguments

x	A neuroim2::ClusteredNeuroVec object
...	Additional arguments passed to underlying write functions
file	Output HDF5 file path
clusters	A ClusteredNeuroVol object defining spatial clusters
mask	Optional LogicalNeuroVol mask (extracted from first element if not provided)
summary	Logical, whether to summarize data by clusters (default: FALSE)
summary_fun	Function to use for summarization (default: mean)
scan_name	Name for the scan (default: "scan_001")
compression	Compression level 0-9 (default: 4)
scan_names	Character vector of scan names (auto-generated if not provided)
overwrite	Logical, whether to overwrite existing file (default: FALSE)
as_multiscan	Logical, whether to create a multi-scan container (default: FALSE)

Details

This method converts a neuroim2::ClusteredNeuroVec to fmristore's HDF5 format. The ClusteredNeuroVec stores time series data per cluster (T x K matrix), which is written as summary data in the HDF5 file.

By default (as_multiscan=FALSE), creates a single scan file and returns an H5ParcellatedScanSummary object. When as_multiscan=TRUE, creates a multi-scan container with one scan and returns an H5ParcellatedMultiScan object.

Value

Invisible NULL or file path, depending on the method

Invisible NULL

Invisible NULL

Invisible file path

Invisible file path

Invisible file path

Invisible NULL

An H5ParcellatedScanSummary object (if as_multiscan=FALSE) or H5ParcellatedMultiScan object (if as_multiscan=TRUE)

Examples

```
## Not run:
# Write full voxel-level data
write_dataset(nvec, file = "output.h5", clusters = cvol)

# Write summarized data (mean per cluster)
write_dataset(nvec, file = "output_summary.h5", clusters = cvol, summary = TRUE)

# Use median for summarization
write_dataset(nvec,
  file = "output_median.h5", clusters = cvol,
  summary = TRUE, summary_fun = median
)

## End(Not run)

## Not run:
# Write multiple scans
scans <- list(nvec1, nvec2, nvec3)
write_dataset(scans, file = "multi_scan.h5", clusters = cvol)

# With custom names and summary
write_dataset(scans,
  file = "multi_summary.h5", clusters = cvol,
  summary = TRUE, scan_names = c("rest", "task1", "task2")
)
```

```
## End(Not run)

## Not run:
# Export existing H5 object to new file
h5_obj <- H5ParcellatedMultiScan("existing.h5")
write_dataset(h5_obj, file = "copy.h5")

## End(Not run)

## Not run:
# Write latent representation
write_dataset(latent_nvec, file = "latent.h5")

## End(Not run)

## Not run:
# Assuming you have a ClusteredNeuroVec object from neuroim2
# cnvec <- neuroim2::ClusteredNeuroVec(...)

# Default: create single scan file
h5_scan <- write_dataset(cnvec, file = "single_scan.h5")

# Create multi-scan container
h5_multi <- write_dataset(cnvec, file = "multi_scan.h5", as_multiscan = TRUE)

## End(Not run)
```

`write_labeled_vec` *Write a NeuroVec to an HDF5 file with NIfTI-like quaternions*

Description

Creates an HDF5 file following a NIfTI-like header layout, storing:

- `/header/dim` => `[4, X, Y, Z, nVols, 1,1,1]`
- `/header/pixdim` => `[0.0, dx, dy, dz, ...]` (Note: qfac stored in /header/qfac)
- `/header/quatern_b,c,d` and `qoffset_x,y,z`
- `/header/qfac` => Quaternion factor (+/-1)
- `/mask` => 3D dataset `[X, Y, Z]` (0/1) at root level
- `/labels` => array of label strings at root level
- `/data/<label>` => 1D array (length = number of nonzero mask voxels) storing the sub-volume values

Usage

```
write_labeled_vec(
  vec,
  mask,
  labels,
  file,
  compression = 4,
  dtype = hdf5r::h5types$H5T_NATIVE_DOUBLE,
  chunk_size = 1024,
  header_values = list(),
  verbose = FALSE
)
```

Arguments

vec	A 4D NeuroVec with dimension [X,Y,Z,nVols].
mask	A LogicalNeuroVol of shape [X,Y,Z] (the same 3D shape as vec).
labels	A character vector of length nVols, labeling each 4D sub-volume.
file	Either a character path to the HDF5 file to create or an open H5File in write mode.
compression	Integer 0-9 for gzip level; default 4.
dtype	An HDF5 data type object (e.g., hdf5r::h5types$H5T_NATIVE_FLOAT). Default is hdf5r::h5types$H5T_NATIVE_DOUBLE.
chunk_size	If non-NULL, the chunk dimension for the 1D datasets. Default is 1024.
header_values	A named list of optional overrides for fields in the header (e.g., list(qform_code=1L, sform_code=2L)). Note that fields derived from 'vec' or 'mask' (like 'dim', 'pixdim', quaternion fields) cannot be overridden here.
verbose	Logical, whether to print verbose messages during processing.

Details

The 4x4 matrix in trans(space(vec)) is passed to matrixToQuatern, which returns a list containing:

- quaternion = c(b, c, d) (the three imaginary parts)
- qfac (+/-1 sign)

This function stores qfac in /header/qfac and sets /header/pixdim[0]=0. We also gather voxel spacing (dx,dy,dz) from spacing(space(vec)) and the origin from origin(space(vec)).

We store a subset of NIfTI-like header fields in the /header group. The user can supply header_values (a named list) to override or augment *some* additional fields (e.g., qform_code=1L). See implementation notes for which fields are protected.

Value

Invisibly returns the open H5File handle containing the written data. The caller should close this handle when finished.

Lifecycle Management

The HDF5 file is opened in write mode and the resulting handle is returned without being automatically closed. **It is the caller's responsibility to close this handle** (via `h5file$close_all()` or `close()`) when finished working with the file.

See Also

`matrixToQuatern` for how the quaternion is derived, `quaternToMatrix` for reconstructing the 4x4, `read_labeled_vec` for reading the file back in.

Examples

```
## Not run:
# Create a simple labeled neuroimaging dataset
vec <- fmristore:::create_minimal_DenseNeuroVec(dims = c(4, 4, 3, 5))

# Create mask with same spatial dimensions
mask <- fmristore:::create_minimal_LogicalNeuroVol(dims = c(4, 4, 3))

# Define labels for each volume
labels <- c("condition_A", "condition_B", "condition_C", "condition_D", "condition_E")

# Write to temporary HDF5 file
temp_file <- tempfile(fileext = ".h5")
h5_handle <- write_labeled_vec(vec, mask, labels, file = temp_file)

# Close the file handle returned by write_labeled_vec
h5_handle$close_all()

# Read it back
labeled_data <- read_labeled_vec(temp_file)
print(labeled_data@labels)

# Clean up
close(labeled_data)
unlink(temp_file)

## End(Not run)
```

---

write_parcellated_experiment_h5

Write Parcellated Experiment Data to HDF5

Description

Writes neuroimaging data structured according to the H5ParcellatedMultiScan specification into an HDF5 file.

This function takes R objects representing the mask, cluster definitions, run-specific data (either full voxel-level or summary time series), and associated metadata, and creates the corresponding HDF5 groups and datasets.

Usage

```
write_parcellated_experiment_h5(
  filepath,
  mask,
  clusters,
  runs_data,
  cluster_metadata = NULL,
  overwrite = FALSE,
  compress = TRUE,
  verbose = TRUE
)

write_clustered_experiment_h5(
  filepath,
  mask,
  clusters,
  runs_data,
  cluster_metadata = NULL,
  overwrite = FALSE,
  compress = TRUE,
  verbose = TRUE
)
```

Arguments

`filepath`	Character string: the path to the HDF5 file to create. If the file exists, it will be overwritten.
`mask`	A ‘LogicalNeuroVol‘ object representing the brain mask.
`clusters`	A ‘ClusteredNeuroVol‘ object containing cluster assignments for voxels within the mask.
`runs_data`	A list where each element represents a single run/scan. Each element must be a list containing: • ‘scan_name‘: (character) Unique identifier for the scan. • ‘type‘: (character) Either "full" or "summary". • ‘data‘: – If ‘type‘ is "full", ‘data‘ must be a list where names are ‘cluster_<cid>‘ (e.g., ‘cluster_1‘, ‘cluster_2‘) and values are matrices ‘[nVoxelsInCluster, nTime]‘ containing the time series for that cluster. – If ‘type‘ is "summary", ‘data‘ must be a single matrix ‘[nTime, nClusters]‘ containing the summary time series. • ‘metadata‘: (Optional) A list of key-value pairs for scan-specific metadata. Can include ‘n_time‘ explicitly, otherwise it’s inferred from data.

cluster_metadata	(Optional) A 'data.frame' containing metadata for the clusters. Must contain at least a column named 'cluster_id' matching the unique IDs in 'clusters'. Other columns will be written as part of a compound dataset.
overwrite	Logical: If 'TRUE', overwrite the file if it exists. Default 'FALSE'.
compress	Logical: If 'TRUE', apply GZIP compression to data arrays. Default 'TRUE'.
verbose	Logical: Print progress messages? Default 'TRUE'.

Value

Invisibly returns 'NULL'. Called for its side effect of creating the HDF5 file.

Examples

```
## Not run:
if (requireNamespace("neuroim2", quietly = TRUE) &&
  requireNamespace("hdf5r", quietly = TRUE) &&
  exists("write_parcellated_experiment_h5", where = "package:fmristore") &&
  !is.null(fmristore:::create_minimal_LogicalNeuroVol) &&
  !is.null(fmristore:::create_minimal_ClusteredNeuroVol)) {
  temp_h5_file <- NULL

  tryCatch({
    # 1. Create a temporary file path
    temp_h5_file <- tempfile(fileext = ".h5")

    # 2. Create minimal mask and clusters using helpers
    mask_vol <- fmristore:::create_minimal_LogicalNeuroVol(dims = c(5L, 5L, 2L))
    # Ensure clusters are within the mask and have some content
    # Create clusters that align with the mask's space
    clust_vol <- fmristore:::create_minimal_ClusteredNeuroVol(
      space = neuroim2::space(mask_vol), # Use mask's space
      mask = mask_vol@.Data, # Use mask's data
      num_clusters = 2L
    )

    # 3. Prepare minimal runs_data
    # Get cluster IDs and number of voxels per cluster from clust_vol
    unique_cids <- sort(unique(clust_vol@clusters[clust_vol@clusters > 0]))
    n_time_run1 <- 10L
    n_time_run2 <- 8L

    # Run 1: Full data type
    run1_data_list <- list()
    if (length(unique_cids) > 0) {
      for (cid in unique_cids) {
        n_vox_in_cluster <- sum(clust_vol@clusters == cid)
        if (n_vox_in_cluster > 0) {
          # Data: nVoxInCluster x nTime
          run1_data_list[[paste0("cluster_", cid)]] <- matrix(
            rnorm(n_vox_in_cluster * n_time_run1),
```

```
        nrow = n_vox_in_cluster,
        ncol = n_time_run1
      )
    }
  }
}

run1 <- list(
  scan_name = "ScanA_Full",
  type = "full",
  data = run1_data_list,
  metadata = list(subject_id = "sub-01", task = "rest", n_time = n_time_run1)
)

# Run 2: Summary data type
# Data: nTime x nClusters
run2_summary_matrix <- matrix(
  rnorm(n_time_run2 * length(unique_cids)),
  nrow = n_time_run2,
  ncol = length(unique_cids)
)
colnames(run2_summary_matrix) <- paste0("cluster_", unique_cids) # Optional: for clarity

run2 <- list(
  scan_name = "ScanB_Summary",
  type = "summary",
  data = run2_summary_matrix,
  metadata = list(subject_id = "sub-01", task = "task", n_time = n_time_run2)
)

runs_data_list <- list(run1, run2)

# 4. Prepare minimal cluster_metadata (optional)
cluster_meta_df <- NULL
if (length(unique_cids) > 0) {
  cluster_meta_df <- data.frame(
    cluster_id = unique_cids,
    name = paste0("Region_", LETTERS[1:length(unique_cids)]),
    size_vox = sapply(unique_cids, function(id) sum(clust_vol@clusters == id))
  )
}

# 5. Call the function
write_parcellated_experiment_h5(
  filepath = temp_h5_file,
  mask = mask_vol,
  clusters = clust_vol,
  runs_data = runs_data_list,
  cluster_metadata = cluster_meta_df,
  overwrite = TRUE,
  verbose = FALSE
)
```

```
    # Verify file was created
    if (file.exists(temp_h5_file)) {
      cat("Successfully wrote parcellated experiment to:", temp_h5_file, "\\n")
      # Optional: Basic check of the HDF5 file structure
      # h5f <- hdf5r::H5File$new(temp_h5_file, mode="r")
      # print(h5f$ls(recursive=TRUE))
      # h5f$close_all()
    }
  }, error = function(e) {
    message("write_parcellated_experiment_h5 example failed: ", e$message)
    if (!is.null(temp_h5_file)) message("Temporary file was: ", temp_h5_file)
  }, finally = {
    # Clean up temporary file
    if (!is.null(temp_h5_file) && file.exists(temp_h5_file)) {
      unlink(temp_h5_file)
    }
  })
} else {
  message("Skipping example: dependencies not available.")
}

## End(Not run)
```

`write_parcellated_scan_h5`

Write Single Parcellated Scan to HDF5

Description

Writes a single parcellated/clustered scan to an HDF5 file. Unlike `write_parcellated_experiment_h5` which creates a multi-scan container, this function creates a file optimized for a single scan.

Usage

```
write_parcellated_scan_h5(
  filepath,
  mask,
  clusters,
  scan_data,
  scan_name = "scan_001",
  data_type = c("summary", "full"),
  scan_metadata = list(),
  cluster_metadata = NULL,
  compress = TRUE,
  verbose = FALSE
)
```

Arguments

filepath	Path to the output HDF5 file
mask	A LogicalNeuroVol object defining the brain mask
clusters	A ClusteredNeuroVol object defining the parcellation
scan_data	Either a named list (for full data) or matrix (for summary data): - For full data: list with cluster names as keys, each containing a voxels x time matrix - For summary data: time x clusters matrix
scan_name	Name for the scan (default: "scan_001")
data_type	Type of data: "full" for voxel-level, "summary" for cluster means
scan_metadata	Optional list of metadata for the scan
cluster_metadata	Optional data.frame with cluster information
compress	Logical, whether to compress the data (default: TRUE)
verbose	Logical, whether to print progress messages (default: FALSE)

Value

An H5ParcellatedScanSummary or H5ParcellatedScan object

---

write_vec,LatentNeuroVec,character,missing,missing-method

Write LatentNeuroVec to HDF5 File

Description

Writes a LatentNeuroVec to an HDF5 file with optional compression.

Usage

```
## S4 method for signature 'LatentNeuroVec,character,missing,missing'
write_vec(x, file_name, compression = 9, nbit = FALSE)
```

Arguments

x	A LatentNeuroVec to write.
file_name	character file path to the output HDF5.
compression	integer in [1..9] specifying compression level (default: 9).
nbit	logical; if TRUE, uses N-bit compression (default: FALSE).

Details

This function saves:

- `basis` matrix (as embedding)
- `loadings` matrix (as spatial basis)
- `offset` vector
- Spatial metadata
- Mask information

all inside an HDF5 file for future loading.

Value

Invisible `NULL`, called for side effects (writes to disk).

See Also

`LatentNeuroVec` (from **fmrilatent**) for the class definition.

Examples

```
## Not run:
library(fmrilatent)
library(fmristore)

# Create a LatentNeuroVec
lvec <- LatentNeuroVec(basis, loadings, space, mask)

# Write to HDF5
temp_file <- tempfile(fileext = ".h5")
write_vec(lvec, temp_file, compression = 6)

## End(Not run)
```

Index

∗ **H5ClusteredIO**
write_parcellated_experiment_h5, 76
∗ **H5Cluster**
[,H5ParcellatedScan,ANY,ANY,ANY-method, 5
[,H5ParcellatedScan,ANY,missing,ANY-method, 6
as.data.frame, 9
as.matrix, 10
H5ParcellatedMultiScan-class, 36
make_run_full, 44
make_run_summary, 46
series, 63
show,H5ParcellatedScan-method, 67
∗ **H5Parcellated**
$,H5ParcellatedMultiScan-method, 8
close, 16
cluster_metadata,H5ParcellatedMultiScan-method, 19
clusters, 20
h5file,H5ParcellatedArray-method, 26
H5ParcellatedMultiScan, 34
H5ParcellatedScan-class, 38
H5ParcellatedScanSummary-class, 40
mask,H5ParcellatedArray-method, 49
matrix_concat, 50
n_scans,H5ParcellatedMultiScan-method, 53
scan_metadata,H5ParcellatedMultiScan-method, 60
scan_names,H5ParcellatedMultiScan-method, 62
series_concat, 64
∗ **datasets**
H5_ATTRS, 24
H5_DSETS, 24
H5_PATHS, 25
[,H5NeuroVec,numeric,numeric,ANY-method *([,H5ParcellatedScan,ANY,missing,ANY-method)*, 6
[,H5NeuroVol,ANY,ANY,ANY-method, 4
[,H5NeuroVol,numeric,missing,ANY-method, 4
[,H5ParcellatedScan,ANY,ANY,ANY-method, 5
[,H5ParcellatedScan,ANY,missing,ANY-method, 6
[,H5ParcellatedScanSummary,ANY,ANY,ANY-method *([,H5ParcellatedScan,ANY,missing,ANY-method)*, 6
[,H5ParcellatedScanSummary,ANY,missing,ANY-method *([,H5ParcellatedScan,ANY,missing,ANY-method)*, 6
[,LabeledVolumeSet,ANY,ANY,ANY-method, 7
[[,H5NeuroVecSeq,ANY-method *([,H5ParcellatedScan,ANY,missing,ANY-method)*, 6
[[,LabeledVolumeSet,character-method *([,H5ParcellatedScan,ANY,missing,ANY-method)*, 6
[[,LabeledVolumeSet,numeric-method *([,H5ParcellatedScan,ANY,missing,ANY-method)*, 6
$,H5ParcellatedMultiScan-method, 8

as.array.LatentNeuroVec, 8
as.data.frame, *5*, *7*, 9, *9, 10*, *36*, *45, 46*, *64*, *67*
as.data.frame,H5ParcellatedScanSummary-method *(as.data.frame)*, 9
as.matrix, *5*, *7*, *9*, 10, *10*, *36*, *45, 46*, *64*, *67*
as.matrix,H5ParcellatedScanSummary-method *(as.matrix)*, 10
as_h5, 10
as_h5,ClusteredNeuroVec-method *(as_h5)*, 10

as_h5,LabeledVolumeSet-method *(as_h5)*, 10
as_h5,LatentNeuroVec-method *(as_h5)*, 10
as_h5,NeuroVec-method *(as_h5)*, 10
as_h5,NeuroVecSeq-method *(as_h5)*, 10
as_h5,NeuroVol-method, 14

basis, 15

close, *8*, 16, *19, 20*, *27, 28*, *32*, *35*, *38*, *40*, *50*, *53*, *59*, *61, 62*, *65*
close,H5NeuroVec-method *(close)*, 16
close,H5NeuroVecSeq-method *(close)*, 16
close,H5NeuroVol-method *(close)*, 16
close,H5ParcellatedMultiScan-method *(close)*, 16
close,H5ParcellatedScan-method *(close)*, 16
close,H5ParcellatedScanSummary-method *(close)*, 16
close,LabeledVolumeSet-method *(close)*, 16
cluster_metadata, 18
cluster_metadata,H5ParcellatedMultiScan-method, 19
ClusteredNeuroVol, *70*
clusters, *8*, *17*, *19*, 20, *27*, *35*, *38*, *40*, *50*, *53*, *61, 62*, *65*
clusters,H5ParcellatedArray-method *(clusters)*, 20
clusters,H5ParcellatedMultiScan-method *(clusters)*, 20
configure_memory_warnings, 21

detect_h5_type, 22
dim, *5*, *7*, *9, 10*, *36*, *45, 46*, *64*, *67*

get_memory_settings, 23

H5_ATTRS, 24
H5_DSETS, 24
H5_PATHS, 25
H5File, *29*, *33*, *75*
h5file, 25
h5file,H5NeuroVecSeq-method *(h5file,H5ParcellatedArray-method)*, 26
h5file,H5ParcellatedArray-method, 26
h5file,H5ParcellatedMultiScan-method *(h5file,H5ParcellatedArray-method)*, 26
h5file,H5ParcellatedScan-method *(h5file,H5ParcellatedArray-method)*, 26
h5file,H5ParcellatedScanSummary-method *(h5file,H5ParcellatedArray-method)*, 26
H5NeuroVec, 27, *27*
H5NeuroVec-class, 29
H5NeuroVecSeq, 30
H5NeuroVecSeq-class, 31
H5NeuroVol, 31, *31*
H5NeuroVol-class, 32
H5ParcellatedMultiScan, *8*, *17*, *19, 20*, *27*, 34, *38*, *40*, *50*, *53*, *61, 62*, *65*
H5ParcellatedMultiScan-class, 36
H5ParcellatedScan, 36, *64*
H5ParcellatedScan-class, 38
H5ParcellatedScanSummary, 38
H5ParcellatedScanSummary-class, 40

LabeledVolumeSet-class, 41
LatentNeuroVec, 42
LatentNeuroVec-io, 43
loadings, 43
LogicalNeuroVol, *75*

make_run_full, *5*, *7*, *9, 10*, *36*, 44, *46*, *64*, *67*
make_run_summary, *5*, *7*, *9, 10*, *36*, *45*, 46, *64*, *67*
map, 47
mask, 48
mask,H5ParcellatedArray-method, 49
mask,H5ParcellatedMultiScan-method *(mask,H5ParcellatedArray-method)*, 49
mask,LatentNeuroVec-method *(mask,H5ParcellatedArray-method)*, 49
matrix_concat, *8*, *17*, *19, 20*, *27*, *35*, *38*, *40*, 50, *50*, *53*, *61, 62*, *65*
matrix_concat,H5ParcellatedMultiScan-method *(matrix_concat)*, 50
matrixToQuatern, *75, 76*

n_scans, 51

n_scans,H5NeuroVecSeq-method
(n_scans,H5ParcellatedMultiScan-method), 53
n_scans,H5ParcellatedMultiScan-method, 53
NeuroVec, *75*
neurovecseq_to_h5, 53

offset, 55

quaternToMatrix, *76*

read_dataset, 56, *70*
read_dataset,character-method, 57
read_dataset,H5File-method
(read_dataset), 56
read_labeled_vec, 58, *76*

scan_metadata, 59
scan_metadata,H5ParcellatedMultiScan-method, 60
scan_names, 61
scan_names,H5NeuroVecSeq-method
(scan_names,H5ParcellatedMultiScan-method), 62
scan_names,H5ParcellatedMultiScan-method, 62
series, *5*, *7*, *9, 10*, *36*, *45, 46*, 63, *67*
series,H5NeuroVec,integer-method
(series), 63
series,H5NeuroVec,matrix-method
(series), 63
series,H5NeuroVec,numeric-method
(series), 63
series,H5ParcellatedScan,matrix-method
(series), 63
series,H5ParcellatedScan,numeric-method
(series), 63
series,H5ParcellatedScanSummary,ANY-method
(series), 63
series_concat, *8*, *17*, *19, 20*, *27*, *35*, *38*, *40*, *50*, *53*, *61, 62*, 64
series_concat,H5ParcellatedMultiScan,numeric-method
(series_concat), 64
show,H5NeuroVec-method, 66
show,H5NeuroVol-method, 66
show,H5ParcellatedMultiScan-method
(show,H5ParcellatedScan-method), 67
show,H5ParcellatedScan-method, 67
show,H5ParcellatedScanSummary-method
(show,H5ParcellatedScan-method), 67
show,LabeledVolumeSet-method, 67

validate_latent_file, 68

write_cluster_result, 69
write_clustered_experiment_h5
(write_parcellated_experiment_h5), 76
write_dataset, *69, 70*, 71
write_dataset,ClusteredNeuroVec-method
(write_dataset), 71
write_dataset,H5NeuroVec-method
(write_dataset), 71
write_dataset,H5NeuroVol-method
(write_dataset), 71
write_dataset,H5ParcellatedMultiScan-method
(write_dataset), 71
write_dataset,LatentNeuroVec-method
(write_dataset), 71
write_dataset,list-method
(write_dataset), 71
write_dataset,NeuroVec-method
(write_dataset), 71
write_labeled_vec, *58, 59*, 74
write_parcellated_experiment_h5, 76
write_parcellated_scan_h5, 80
write_vec,LatentNeuroVec,character,missing,missing-method, 81