Package 'bidser'

May 7, 2026

Version 0.2.0

Title Work with 'BIDS' (Brain Imaging Data Structure) Projects

Description Tools for working with 'BIDS' (Brain Imaging Data Structure) formatted neuroimaging datasets. The package provides functionality for reading and querying 'BIDS'-compliant projects, creating mock 'BIDS' datasets for testing, and extracting preprocessed data from 'fMRIPrep' derivatives. It supports searching and filtering 'BIDS' files by various entities such as subject, session, task, and run to streamline neuroimaging data workflows. See Gorgolewski et al. (2016) <doi:10.1038/sdata.2016.44> for the 'BIDS' specification.

License MIT + file LICENSE

Encoding UTF-8

ByteCompile true

RoxygenNote 7.3.3

Imports stringr, data.tree, neuroim2, tidyselect, dplyr, assertthat, crayon, fs, jsonlite, magrittr, purrr, readr, rlang, stringdist, tibble, tidyr, httr, rio

Suggests ggplot2, plotly, patchwork, viridis, scales, knitr, rmarkdown, testthat, covr, lintr, gluedown, RNifti, future, future.apply

VignetteBuilder knitr

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

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

Config/Needs/website albersdown

NeedsCompilation no

Author Bradley Buchsbaum [aut, cre] (ORCID: <https://orcid.org/0000-0002-1108-4866>)

Maintainer Bradley Buchsbaum <brad.buchsbaum@gmail.com>

Repository CRAN

Date/Publication 2026-02-19 20:20:12 UTC

Contents

`anat_parser` *Anatomical parser constructor*

Description

Anatomical parser constructor

Usage

```
anat_parser()
```

Value

An anatomical BIDS parser object for parsing anatomical files

Examples

```
# Create an anatomical parser
parser <- anat_parser()

# Parse an anatomical file
result <- parse(parser, "sub-01_T1w.nii.gz")
```

`bids_check_compliance` *Basic BIDS Compliance Checks*

Description

This function performs a simple, lightweight check of common BIDS requirements:

- Checks that `participants.tsv` and `dataset_description.json` exist at the root.
- Ensures all subject directories begin with `sub-`.
- If sessions are present, ensures that session directories begin with `ses-`.

Usage

```
bids_check_compliance(x)

bids_check_compliance(x)
```

Arguments

`x`	A `bids_project` object.

Details

Note: This is not a full BIDS validator. For complete validation, use the official BIDS validator.

Value

A list with compliance check results

A list with:

- `passed` (logical): TRUE if all checks passed, FALSE otherwise.
- `issues` (character vector): Descriptions of any issues found.

Examples

```
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  compliance <- bids_check_compliance(proj)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

bids_heatmap *Create a specialized heatmap visualization of BIDS data*

Description

This function creates a heatmap visualization of a BIDS project, where the x-axis represents subjects and the y-axis represents tasks by run. Each cell in the heatmap is colored by file size, providing an intuitive view of data completeness and size distribution across the project. This is particularly useful for quality control and identifying missing data.

Usage

```
bids_heatmap(
  x,
  interactive = TRUE,
  color_scheme = "viridis",
  file_type = "func",
  highlight_missing = TRUE,
  text_size = 2.5,
  rotate_labels = TRUE
)
```

Arguments

x	A bids_project object
interactive	Logical. Whether to create an interactive plot (default TRUE)
color_scheme	Character. Name of the color palette to use (default "viridis")
file_type	Character. Type of files to visualize (default "func")
highlight_missing	Logical. Whether to highlight missing data points (default TRUE)
text_size	Numeric. Size of text labels (default 2.5)
rotate_labels	Logical. Whether to rotate the axis labels (default TRUE)

Value

A plot object (ggplot2 or plotly depending on interactive parameter)

Examples

```
# Create a basic interactive heatmap for a BIDS dataset
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  bids_heatmap(proj)

  # Create a static heatmap with custom settings
```

```
  bids_heatmap(proj,
               interactive = FALSE,
               color_scheme = "plasma",
               text_size = 3,
               rotate_labels = FALSE)

  # Visualize anatomical data with missing data highlighted
  bids_heatmap(proj,
               file_type = "anat",
               highlight_missing = TRUE,
               color_scheme = "magma")

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`bids_parser` *BIDS filename parsers using regex*

Description

These functions create parsers for different types of BIDS files using regex-based pattern matching instead of parser combinators. Create a parser for a generic BIDS file

Usage

```
bids_parser()
```

Details

This parser tries to match against various known parsers (anat, func, fmriprep anat/func).

Value

A BIDS parser object that can parse various types of BIDS files

Examples

```
# Create a generic BIDS parser
parser <- bids_parser()

# Parse different types of files
anat_result <- parse(parser, "sub-01_T1w.nii.gz")
func_result <- parse(parser, "sub-01_task-rest_bold.nii.gz")
prep_result <- parse(parser, "sub-01_task-rest_space-MNI152NLin2009cAsym_desc-preproc_bold.nii.gz")
```

bids_project *Create a BIDS Project Object*

Description

This function creates a BIDS project object from a directory containing BIDS-formatted neuroimaging data. It can optionally load preprocessed derivatives from fMRIPrep. The function validates the basic BIDS structure and provides methods for accessing raw and preprocessed data, querying subjects, sessions, and tasks, reading event files, and checking BIDS compliance.

Usage

```
bids_project(path = ".", fmriprep = FALSE, prep_dir = "derivatives/fmriprep")
```

Arguments

path	Character string. The file path to the root of the BIDS project. Defaults to the current directory (".).
fmriprep	Logical. Whether to load the fMRIPrep derivatives folder hierarchy. Defaults to FALSE.
prep_dir	Character string. The location of the fMRIPrep subfolder relative to the derivatives directory. Defaults to "derivatives/fmriprep".

Value

A bids_project object representing the BIDS project structure. The object provides methods for:

- Accessing raw and preprocessed data files
- Querying subjects, sessions, and tasks
- Reading event files and confound regressors
- Checking BIDS compliance
- Extracting metadata from file names Returns NULL if the directory does not contain a valid BIDS dataset.

Examples

```
# Create a BIDS project
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Get all functional scans
  all_scans <- func_scans(proj)

  # Get scans for specific subjects
  sub_scans <- func_scans(proj, subid="0[123]")
```

```
  # Get scans for a specific task
  task_scans <- func_scans(proj, task="rest")

  # Get scans from specific runs
  run_scans <- func_scans(proj, run="0[123]")

  # Combine multiple filters
  filtered_scans <- func_scans(proj,
                               subid="01",
                               task="rest",
                               run="01")

  # Get relative paths instead of full paths
  rel_scans <- func_scans(proj, full_path=FALSE)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`bids_subject` *Access a single subject from a BIDS project*

Description

`bids_subject` returns a lightweight interface with helper functions for retrieving data associated with one subject.

`bids_subject` returns a lightweight facade that exposes convenience functions to work with all data associated with one subject within a BIDS project.

This function extracts a single subject's data from a BIDS project, creating a new BIDS project object containing only that subject's files and metadata.

Usage

```
bids_subject(x, subid, ...)

bids_subject.bids_project(x, subid, ...)

bids_subject(x, subid, ...)
```

Arguments

`x`	A `bids_project` object.
`subid`	Character string. The subject ID to extract (without the "sub-" prefix).
`...`	Additional arguments (not currently used).

Value

A list of helper functions for the subject.

A list containing subject-specific helper functions. Each function automatically filters results for the specified subject. The returned object contains the following callable functions:

`events(...)` Returns nested tibble with event data for this subject. Equivalent to `read_events(project, subid = "XX", ...)`. Additional arguments (task, session, run, nest, etc.) can be passed.

`event_files(...)` Returns character vector of event file paths for this subject. Equivalent to `event_files(project, subid = "XX", ...)`. Additional arguments (task, session, run, full_path, etc.) can be passed.

`scans(...)` Returns character vector of functional scan file paths for this subject. Equivalent to `func_scans(project, subid = "XX", ...)`. Additional arguments (task, session, run, kind, full_path, etc.) can be passed.

`confounds(...)` Returns confound data for this subject (requires fMRIPrep derivatives). Equivalent to `read_confounds(project, subid = "XX", ...)`. Additional arguments (task, session, run, cvars, npcs, etc.) can be passed.

`preproc_scans(...)` Returns preprocessed scan paths for this subject (requires fMRIPrep derivatives). Equivalent to `preproc_scans(project, subid = "XX", ...)`. Additional arguments (task, session, run, space, variant, etc.) can be passed.

`brain_mask(...)` Creates brain mask for this subject (requires fMRIPrep derivatives). Equivalent to `brain_mask(project, subid = "XX", ...)`. Additional arguments (thresh, etc.) can be passed.

A new `bids_project` object containing only the specified subject's data. Returns NULL if the subject is not found in the project.

Examples

```
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  subj <- bids_subject(proj, "01")
  subj$events()
  subj$scans()

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})


tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Create subject interface for subject 01
  subj <- bids_subject(proj, "01")
```

```
  # Get functional scan paths for this subject
  scan_paths <- subj$scans()
  print(paste("Subject 01 has", length(scan_paths), "functional scans"))

  # Get event file paths for this subject
  event_paths <- subj$event_files()
  print(paste("Subject 01 has", length(event_paths), "event files"))

  # Read event data for this subject
  event_data <- subj$events()
  print("Event data structure:")
  print(event_data)

  # You can still pass additional filtering arguments
  # For example, get only specific tasks:
  task_scans <- subj$scans(task = "balloonanalogrisktask")

  # Dataset cache is intentionally retained for performance.
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})


# Create a subject interface
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Create subject interface for subject 01
  subj <- bids_subject(proj, "01")

  # Use the helper functions
  scans <- subj$scans()
  events <- subj$event_files()
  print(paste("Subject 01:", length(scans), "scans,", length(events), "events"))

  # Dataset cache is intentionally retained for performance.
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`bids_summary` *Summarize a BIDS dataset*

Description

Provides a quick summary of dataset statistics, including:

- Number of subjects

- Number of sessions (if applicable)
- Available tasks and the number of runs per task
- Total number of runs

Usage

```
bids_summary(x)

bids_summary(x)
```

Arguments

x	A bids_project object.

Value

A list containing summary statistics about the BIDS dataset

A list with summary information:

- n_subjects: number of participants
- n_sessions: number of sessions (if any), otherwise NULL
- tasks: a data frame with task and n_runs columns
- total_runs: total number of runs across the dataset

Examples

```
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  summary <- bids_summary(proj)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

bids_transform *Apply a transformation to BIDS files*

Description

This function orchestrates the process of selecting files from a BIDS project, applying a transformation to each file, and saving the output in a new BIDS derivative directory. It leverages the existing bidser parsing and search infrastructure.

Usage

```
bids_transform(x, transformer, pipeline_name, ...)
```

Arguments

x	A bids_project object.
transformer	A function that performs the transformation. It must take the input file path and return the output file path. The transformer is responsible for creating the output file.
pipeline_name	The name for the new derivative pipeline.
...	Additional arguments passed to search_files to select files (e.g., subid = "01", task = "rest").

Value

A character vector of paths to the newly created files.

Examples

```
tryCatch({
  ds_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds_path)

  # Create a simple transformer that adds a description
  add_desc_transformer <- function(infile) {
    entities <- encode(basename(infile))
    entities$desc <- if (is.null(entities$desc)) "smooth6mm" else
                     paste(entities$desc, "smooth6mm", sep="")

    # Generate new filename
    new_name <- decode_bids_entities(entities)
    outfile <- file.path(dirname(infile), new_name)

    # For demo, just copy the file (real transformer would process it)
    file.copy(infile, outfile)
    return(outfile)
  }

  # Apply transformation to functional files for subject 01
  new_files <- bids_transform(proj, add_desc_transformer, "smoothed",
                              subid = "01", suffix = "bold.nii.gz")
  print(length(new_files))

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

`brain_mask` *Retrieve a brain mask for a subject*

Description

This convenience function wraps `create_preproc_mask()` and returns a brain mask volume for a given subject.

Usage

```
brain_mask(x, subid, ...)

## S3 method for class 'bids_project'
brain_mask(x, subid, ...)
```

Arguments

`x`	A bids_project object
`subid`	A regular expression pattern to match subject IDs
`...`	Additional arguments passed to methods

Value

A logical mask volume

Examples

```
# Download and load a BIDS project with fMRIPrep derivatives
tryCatch({
  ds001_deriv_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds001_deriv_path, fmriprep=TRUE)
  mask <- brain_mask(proj, subid="01")

  # Create mask for multiple subjects
  multi_mask <- brain_mask(proj, subid=".*")

  # Clean up
  unlink(ds001_deriv_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires derivatives dataset: ", e$message)
})
```

`build_subject_graph` *Build Subject Graph Structure*

Description

Creates a structured list or tibble containing all available data for a single subject, organized by data type. This provides a comprehensive view of all available files for a subject, useful for batch processing and pipeline ingestion.

Usage

```
build_subject_graph(x, subid, session = ".*", flatten = FALSE, ...)

## S3 method for class 'bids_project'
build_subject_graph(x, subid, session = ".*", flatten = FALSE, ...)

## S3 method for class 'mock_bids_project'
build_subject_graph(x, subid, session = ".*", flatten = FALSE, ...)
```

Arguments

`x`	A `bids_project` or `mock_bids_project` object.
`subid`	Subject identifier (with or without `sub-` prefix).
`session`	Optional session filter. Default `".*"` matches all sessions.
`flatten`	Logical. If `FALSE` (default), return a nested list structure. If `TRUE`, return a flat tibble with columns for file_type and metadata.
`...`	Additional arguments passed to underlying query functions.

Value

If `flatten = FALSE` (default), a named list with class `bids_subject_graph`:

subid Subject identifier (without "sub-" prefix)

sessions Character vector of available sessions

epi Named list of preprocessed EPI file paths, keyed by task.run

anat List with t1w and masks sublists

transforms Named list of transform files, keyed by from_to_to format

surfaces Nested list by space, then hemisphere (L/R)

confounds Character vector of confound file paths

If `flatten = TRUE`, a tibble with columns:

file_type Type of file (epi, anat, transform, surface, confound)

path File path

subid, session, task, run, space, hemi, from, to BIDS metadata

Examples

```
# Build subject graph
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep = TRUE)

  # Get nested structure
  graph <- build_subject_graph(proj, "01")
  names(graph)

  # Get flat tibble
  flat <- build_subject_graph(proj, "01", flatten = TRUE)
  head(flat)

  # Clean up
  unlink(ds_path, recursive = TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

check_func_scans *Check Functional Scans in a BIDS Project*

Description

This function performs a comprehensive inspection of functional scans within a BIDS project, providing detailed summaries of scan counts and file sizes per subject and task. It helps identify potential issues such as missing scans, inconsistent file sizes, or unexpected variations in the data.

Usage

```
check_func_scans(x)
```

Arguments

x	A `bids_project` object created by `bids_project()`.

Value

A list containing:

- `scans`: A tibble with details of all functional scans, including:
  - Subject ID
  - Task name
  - Run number
  - File size
  - Full file path

- `tasklist`: A vector of unique tasks found in the project
- `scans_per_subject`: A summary tibble showing the number of scans per subject

If multiple tasks are present, also includes:

- `scans_per_task`: Summary of scan counts by task
- `scans_per_task_subject`: Summary of scan counts by subject and task
- `size_per_task`: Tibble with file size statistics by task

If only one task is present:

- `size_per_subject`: Tibble with file size statistics by subject

Examples

```
# Check functional scans in a BIDS dataset
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  scan_check <- check_func_scans(proj)
  print(scan_check)

  # Filter for specific subjects
  sub01_check <- check_func_scans(proj, subid="01")

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`clear_example_bids_cache` *Clear Example BIDS Dataset Cache*

Description

Clears the session-level cache of downloaded example BIDS datasets. This can be useful to free up memory or force re-download of datasets.

Usage

```
clear_example_bids_cache()
```

Value

Invisible NULL

Examples

```
# Clear the cache
clear_example_bids_cache()
```

`confound_files` *Get confound files from a BIDS project*

Description

This function retrieves a vector of confound files from a BIDS project that match specified criteria. Confound files in BIDS derivatives (typically from fMRIPrep) contain nuisance variables that can be used for denoising fMRI data, such as motion parameters, physiological signals, and other noise components.

Searches the mock BIDS structure for files matching typical confound file patterns (e.g., `*_confounds*.tsv`, `*_regressors*.tsv`, `*_timeseries*.tsv`) within the derivatives directory.

Usage

```
confound_files(x, ...)

## S3 method for class 'bids_project'
confound_files(x, subid = ".*", task = ".*", session = ".*", ...)

## S3 method for class 'mock_bids_project'
confound_files(
  x,
  subid = ".*",
  task = ".*",
  session = ".*",
  run = ".*",
  full_path = FALSE,
  ...
)
```

Arguments

`x`	A `mock_bids_project` object.
`...`	Additional arguments passed to `search_files`.
`subid`	Regex pattern for subject IDs. Default `".*"`.
`task`	Regex pattern for task names. Default `".*"`.
`session`	Regex pattern for session IDs. Default `".*"`.
`run`	Regex pattern for run indices. Default `".*"`.
`full_path`	If `TRUE`, return full paths (prefixed with `x$path`). If `FALSE` (default), return relative paths.

Details

This function assumes confound files reside in the derivatives path specified by `x$prep_dir` and were defined in the `file_structure` passed to `create_mock_bids` with `fmriprep=TRUE`.

Value

A character vector of file paths to confound files matching the specified criteria. If no matching files are found, returns NULL.

A character vector of file paths

A character vector of relative or full paths to potential confound files, or `NULL` if none are found.

Examples

```
# Get all confound files from a BIDS project with fMRIPrep derivatives
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep=TRUE)
  conf_files <- confound_files(proj)

  # Get confound files for specific subjects and tasks
  confound_files(proj, subid="sub-01", task="balloonanalogrisktask")

  # Clean up
  unlink(ds_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})

# Setup mock project with a derivative confound file
participants_df <- tibble::tibble(participant_id = "01")
file_structure_df <- tibble::tribble(
  ~subid, ~session, ~datatype, ~task, ~run, ~suffix, ~fmriprep, ~desc,
  "01",   NA,       "func",    "taskA", "01",
  "desc-confounds_timeseries.tsv", TRUE, "confounds"
)
mock_proj <- create_mock_bids("ConfoundMock", participants_df, file_structure_df)

# Find confound files
confound_files(mock_proj)

# Find for specific subject
confound_files(mock_proj, subid="01")
```

`confound_set` *Convenience confound sets for fMRIPrep*

Description

Provides predefined, version-robust groups of confound variable names as described in the fMRIPrep documentation. These sets abstract over naming changes between fMRIPrep releases via the internal alias resolver used by `read_confounds()`.

Usage

```
confound_set(name, n = NULL)
```

Arguments

`name`	Character. The name of the convenience set (see list above).
`n`	Optional integer used by CompCor sets to limit the number of components (e.g., first 5 or 6). Ignored for other sets.

Details

In addition to exact names, wildcard patterns are supported by the confound resolver:

- `prefix*` selects all columns that start with `prefix` (e.g., `cosine_*`, `motion_outlier_*`, `a_comp_cor_*`).
- `prefix*[N]` selects the first `N` matches (e.g., `a_comp_cor_*[6]`).
- Suffix combinations such as `_derivative1`, `_power2`, and `_derivative1_power2` are resolved across both old and new base names (e.g., `trans_x_derivative1` also finds `X_derivative1` if present).

Available sets (case-insensitive):

- `"motion6"`: 6 rigid-body motion parameters: `trans_x`, `trans_y`, `trans_z`, `rot_x`, `rot_y`, `rot_z`.
- `"motion12"`: `motion6` + first temporal derivatives (adds `*_derivative1`).
- `"motion24"`: `motion12` + quadratic terms of base and derivatives (adds `*_power2` and `*_derivative1_power2`).
- `"global3"`: global signals: `csf`, `white_matter`, `global_signal`.
- `"9p"`: `motion6` + `global3` (9 parameters total).
- `"36p"`: `motion24` + `global3` plus their derivatives and quadratics (i.e., the canonical 36-parameter set).
- `"acompcor"`: anatomical CompCor components (`a_comp_cor_*`). Use `n` to cap the number of components retained, e.g., `n = 6` -> `a_comp_cor_*[6]`.
- `"tcompcor"`: temporal CompCor components (`t_comp_cor_*`). Supports `n` as above.
- `"compcor"`: both anatomical and temporal CompCor (applies `n` to each family if provided).
- `"cosine"`: discrete cosine-basis regressors (matches both `cosine_*` and `cosine*`).
- `"outliers"`: outlier/censoring covariates including `framewise_displacement`, `rmsd` (if present), `motion_outlier_*`, and `non_steady_state_outlier*`.
- `"dvars"`: DVARS family: `dvars`, `std_dvars`, `non_std_dvars`, `vx_wisestd_dvars` (resolved to whichever names exist in your dataset).
- `"fd"`: framewise displacement only (`framewise_displacement`).

Value

A character vector of confound variable names and/or wildcard tokens that can be passed to `read_confounds(..., cvars = confound_set(...))`.

Examples

```
# Common usage: 24-parameter motion set
confound_set("motion24")

# 36-parameter model (Satterthwaite/Friston-style)
confound_set("36p")

# First 6 anatomical CompCor components
confound_set("acompcor", n = 6)

# All cosine regressors and outlier indicators
confound_set("cosine")
confound_set("outliers")
```

`confound_strategy` *Confound denoising strategies*

Description

Creates a structured confound strategy object that specifies which variables to reduce via PCA and which to keep as-is. Pass the result directly to `read_confounds(..., cvars = confound_strategy(...))`.

Usage

```
confound_strategy(
  name = NULL,
  pca_vars = NULL,
  raw_vars = NULL,
  perc_var = -1,
  npcs = -1
)
```

Arguments

`name`	Character. Name of a predefined strategy (see above), or `NULL` for a custom strategy.
`pca_vars`	Character vector of confound names/wildcards to include in PCA reduction. Ignored when `name` is specified.
`raw_vars`	Character vector of confound names/wildcards to keep without reduction. Ignored when `name` is specified.
`perc_var`	Numeric. Percentage of variance to retain from PCA (default -1, meaning use `npcs` instead).
`npcs`	Integer. Number of PCs to retain (default -1, meaning use `perc_var` instead).

Details

When a strategy is passed to `read_confounds`, the function:

1. Selects the `pca_vars` columns and reduces them via PCA (retaining `perc_var\`
2. Selects the `raw_vars` columns and keeps them unchanged.
3. Column-binds the PCA scores with the raw columns.

Available named strategies:

`"pcabasic80"` PCA over motion24 + aCompCor + tCompCor + CSF + white matter, retaining 80\
Discrete cosine regressors are appended un-reduced.

Value

A `confound_strategy` object (S3 class) that can be passed as the `cvars` argument to `read_confounds()`.

Examples

```
# Named strategy
confound_strategy("pcabasic80")

# Custom strategy: PCA motion + compcor to 5 PCs, keep cosine regressors
confound_strategy(
  pca_vars = c(confound_set("motion24"), confound_set("compcor")),
  raw_vars = confound_set("cosine"),
  npcs = 5
)
```

`create_mock_bids` *Create a Mock BIDS Project Object*

Description

Generates an in-memory representation of a BIDS project, suitable for testing and demonstration without requiring actual data files. Can optionally create a "stub" directory structure on disk.

Usage

```
create_mock_bids(
  project_name,
  participants,
  file_structure,
  dataset_description = NULL,
  event_data = list(),
  confound_data = list(),
  create_stub = FALSE,
  stub_path = NULL,
  prep_dir = "derivatives/fmriprep"
)
```

Arguments

project_name	A character string for the project name.
participants	Either a data.frame mirroring participants.tsv content (must include 'participant_id') or a character vector of participant IDs (e.g., c("01", "02")). If IDs are given, a minimal part_df is created.
file_structure	A data.frame or tibble defining the files in the mock structure. Each row represents a file. Required columns: subid, datatype, suffix. Optional BIDS entity columns: session, task, run, acq, rec, dir, space, desc, etc. Must also include a logical column fmriprep indicating if the file belongs in the derivatives directory specified by prep_dir.
dataset_description	A list representing the dataset_description.json content. Defaults to a minimal valid description.
event_data	A named list where names are the *relative paths* of events.tsv files (e.g., "sub-01/func/sub-01_task-A_run-1_events.tsv") and values are the corresponding tibble or data.frame content for those files. These paths must correspond to files defined in file_structure with a suffix like "events.tsv".
confound_data	A named list where names are relative paths of confound TSV files within the derivatives directory and values are their tibble or data.frame content. Paths must match files defined in file_structure.
create_stub	Logical (default FALSE). If TRUE, write a stub BIDS directory structure to disk at stub_path. Zero-byte files are created except for participants.tsv, dataset_description.json, and events.tsv files specified in event_data.
stub_path	Character string, the path where the stub directory will be created. Required if create_stub = TRUE.
prep_dir	Character string, the path relative to the root for derivatives (default "derivatives/fmriprep"). This path structure will be used both in the internal data.tree and for stub creation.

Value

An object of class mock_bids_project.

Examples

```
# --- Example Setup ---
participants_df <- tibble::tibble(participant_id = c("01", "02"), age = c(25, 30))

file_structure_df <- tibble::tribble(
  ~subid, ~session, ~datatype, ~task,   ~run, ~suffix,                ~fmriprep, ~desc,
  "01",   NA,       "anat",    NA,      NA,   "T1w.nii.gz",            FALSE,     NA,
  "01",   NA,       "func",    "taskA", "01", "bold.nii.gz",           FALSE,     NA,
  "01",   NA,       "func",    "taskA", "01", "events.tsv",            FALSE,     NA,
  "02",   "test",   "anat",    NA,      NA,   "T1w.nii.gz",            FALSE,     NA,
  "02",   "test",   "func",    "taskA", "01", "bold.nii.gz",           FALSE,     NA,
  "02",   "test",   "func",    "taskA", "01", "events.tsv",            FALSE,     NA,
  # Example derivative
```

```
  "01",   NA,      "func",    "taskA", "01", "preproc_bold.nii.gz",    TRUE,     "preproc"
)

# Define event data (paths must match generated structure)
event_data_list <- list()
event_data_list[["sub-01/func/sub-01_task-taskA_run-01_events.tsv"]] <- tibble::tibble(
  onset = c(1.0, 5.0), duration = c(0.5, 0.5), trial_type = c("condA", "condB")
)
event_data_list[["sub-02/ses-test/func/sub-02_ses-test_task-taskA_run-01_events.tsv"]] <-
  tibble::tibble(
    onset = c(1.5, 5.5), duration = c(0.5, 0.5), trial_type = c("condC", "condD")
)

# Create the mock project (in memory only)
mock_proj <- create_mock_bids(
  project_name = "MockTaskA",
  participants = participants_df,
  file_structure = file_structure_df,
  event_data = event_data_list
)

# Create the mock project and write stubs
mock_proj_stub <- create_mock_bids(
  project_name = "MockTaskA_stub",
  participants = c("01", "02"), # Example using just IDs
  file_structure = file_structure_df,
  event_data = event_data_list,
  create_stub = TRUE,
  stub_path = tempdir() # Use a temporary directory for example
)

# --- Using the Mock Project ---
print(mock_proj)
print(participants(mock_proj))
print(tasks(mock_proj))
print(sessions(mock_proj)) # Should return "test"

print(func_scans(mock_proj, subid = "01"))
print(event_files(mock_proj, subid = "02", session = "test"))

# Read the injected event data
events_sub1 <- read_events(mock_proj, subid = "01")
print(events_sub1)
if (nrow(events_sub1) > 0) print(tidyr::unnest(events_sub1, cols = data))

# Search for derivatives
print(search_files(mock_proj, suffix = "preproc_bold.nii.gz"))

# Check stub directory (if created)
stub_files <- list.files(mock_proj_stub$path, recursive = TRUE)
print(head(stub_files))

# Read one injected stub event file if present
```

```
stub_event_path <- file.path(mock_proj_stub$path, names(event_data_list)[1])
if (file.exists(stub_event_path)) {
  print(readLines(stub_event_path, n = 1))
}

# Cleanup is intentionally omitted in this example.
```

`create_preproc_mask` *Create a preprocessing mask from BIDS data*

Description

Create a preprocessing mask from BIDS data

Usage

```
create_preproc_mask(x, subid, thresh = 0.99, ...)
```

Arguments

`x`	A bids_project object
`subid`	A regular expression pattern to match subject IDs
`thresh`	Threshold value for mask creation (default: 0.99)
`...`	Additional arguments passed to methods

Value

A logical mask volume

Examples

```
# Download and load a BIDS project with fMRIPrep derivatives
tryCatch({
  ds001_deriv_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds001_deriv_path, fmriprep=TRUE)
  mask <- create_preproc_mask(proj, subid=".*")

  # Create mask for single subject
  sub01_mask <- create_preproc_mask(proj, subid="01")

  # Clean up
  unlink(ds001_deriv_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires derivatives dataset: ", e$message)
})
```

`create_preproc_mask.bids_project` *Create a binary brain mask from preprocessed scans*

Description

This function creates a binary brain mask from preprocessed functional scans in a BIDS project. It searches for BOLD brain mask files in the fMRIPrep derivatives directory (i.e., files in the `func/` folder matching the pattern `*_desc-brain_mask.nii.gz` or the older `*_brainmask.nii.gz`), reads them with neuroim2, averages them, and thresholds the result to produce a consensus binary mask.

Usage

```
## S3 method for class 'bids_project'
create_preproc_mask(
  x,
  subid,
  thresh = 0.99,
  task = ".*",
  space = ".*",
  mask_kinds = c("brainmask", "mask"),
  ...
)
```

Arguments

`x`	A `bids_project` object with fMRIPrep derivatives.
`subid`	Regular expression to match subject IDs (e.g., `"01"` for subject 01, `".*"` for all subjects).
`thresh`	Threshold value between 0 and 1 (default 0.99). Voxels below this value in the averaged mask are excluded. Higher values produce more conservative masks.
`task`	Regular expression for task filtering. Defaults to `".*"` (any task). Because functional masks always carry a `task` entity, this also implicitly excludes anatomical masks which lack it.
`space`	Regular expression for output-space filtering (e.g., `"MNI152NLin2009cAsym"`). Defaults to `".*"` (all spaces). When masks from multiple spaces are found the function stops with an error because their dimensions are incompatible.
`mask_kinds`	Character vector of BIDS suffixes to search. Defaults to both `"brainmask"` (older fMRIPrep) and `"mask"` with `desc="brain"` (fMRIPrep >= 21).
`...`	Additional arguments passed to `search_files` for finding mask files (e.g., `session`, `run`).

Details

The search is restricted to **functional** brain masks by requiring the `task` BIDS entity (anatomical masks do not carry `task`). When masks from multiple output spaces are discovered the function raises an error; pass a specific `space` value to disambiguate.

Value

A logical mask volume (`LogicalNeuroVol`) suitable for use with preprocessed functional data.

Examples

```
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep=TRUE)

  # Mask for one subject in a specific space
  mask <- create_preproc_mask(proj, subid="01",
                              space="MNI152NLin2009cAsym")

  # Consensus mask across all subjects / runs
  all_mask <- create_preproc_mask(proj, subid=".*",
                                  space="MNI152NLin2009cAsym")

  # Restrict to a single task
  task_mask <- create_preproc_mask(proj, subid=".*",
                                  task="balloonanalogrisktask",
                                  space="MNI152NLin2009cAsym")

  unlink(ds_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires derivatives dataset: ", e$message)
})
```

---

create_preproc_mask.mock_bids_project

Create Preprocessing Mask (Mock Implementation)

Description

This function is not implemented for `mock_bids_project` objects as they do not contain actual image data required to create a mask.

Usage

```
## S3 method for class 'mock_bids_project'
create_preproc_mask(x, ...)
```

Arguments

`x`	A `mock_bids_project` object.
`...`	Arguments (ignored).

Value

Throws an error indicating the function is not applicable to mock objects.

Examples

```
mock <- create_mock_bids("Test", c("01"), tibble::tibble(
  subid = "01", datatype = "func",
  suffix = "bold.nii.gz", fmriprep = FALSE
))
try(create_preproc_mask(mock))
```

---

create_smooth_transformer

Create a simple smoothing transformer

Description

This creates a transformer function that adds a smoothing description to BIDS filenames. This is a lightweight example - real implementations would perform actual image processing.

Usage

```
create_smooth_transformer(fwhm, suffix_pattern = "bold\\.nii")
```

Arguments

fwhm	The smoothing FWHM to add to the description.
suffix_pattern	Optional regex pattern to match specific file types.

Value

A transformer function for use with bids_transform.

Examples

```
# Create a smoothing transformer
smooth_6mm <- create_smooth_transformer(6)

# Apply it to a toy BIDS-like file path
in_dir <- tempdir()
out_dir <- tempdir()
infile <- file.path(in_dir, "sub-01_task-rest_bold.nii.gz")
file.create(infile)
new_file <- smooth_6mm(infile, out_dir)
basename(new_file)
unlink(infile)
unlink(new_file)
```

decode_bids_entities *Decode BIDS entities back into a filename*

Description

This function reconstructs a BIDS filename from parsed entities, using the standard BIDS entity ordering.

Usage

```
decode_bids_entities(entities)
```

Arguments

entities	A named list of BIDS entities (from encode).

Value

A character string representing the BIDS filename.

Examples

```
# Parse a filename and reconstruct it
entities <- encode("sub-01_task-rest_run-01_bold.nii.gz")
filename <- decode_bids_entities(entities)
print(filename)

# Modify entities and create new filename
entities$desc <- "smooth6mm"
new_filename <- decode_bids_entities(entities)
print(new_filename)
```

encode *Encode a string into a BIDS key-value list*

Description

This function parses a BIDS filename and extracts its components into a key-value list. It understands standard BIDS entities like subject, session, task, run, etc.

Usage

```
encode(x, ...)

## S3 method for class 'character'
encode(x, ...)
```

Arguments

x	The filename string to encode
...	Additional arguments passed to methods

Value

A list of key-value pairs extracted from the filename

Examples

```
# Encode an anatomical file
encode("sub-01_T1w.nii.gz")

# Encode a functional file
encode("sub-01_task-rest_run-01_bold.nii.gz")

# Encode a file with session information
encode("sub-01_ses-pre_task-rest_run-01_bold.nii.gz")
```

---

event_files *Get event files from a BIDS project*

Description

This function retrieves a vector of event files (events.tsv) from a BIDS project that match specified criteria. Event files in BIDS contain trial information for task-based functional MRI data, including onset times, durations, and trial types.

Finds event files matching the given subject, task, run, and session criteria.

Usage

```
event_files(x, ...)

## S3 method for class 'bids_project'
event_files(
  x,
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  full_path = TRUE,
  ...
)

## S3 method for class 'mock_bids_project'
event_files(
  x,
```

```
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  full_path = TRUE,
  ...
)
```

Arguments

x	A mock_bids_project object
...	Additional arguments passed to internal functions
subid	Regex to match subject IDs (default: ".*")
task	Regex to match tasks (default: ".*")
run	Regex to match runs (default: ".*")
session	Regex to match sessions (default: ".*")
full_path	If TRUE, return full paths of files (default: TRUE)

Value

A character vector of file paths to event files matching the specified criteria. If no matching files are found, returns NULL.

A character vector of file paths to event files. If no matching files are found, returns an empty character vector.

Examples

```
# Get all event files from a BIDS project
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  event_files(proj)

  # Get event files for specific subjects and tasks
  if (length(participants(proj)) > 0) {
    event_files(proj, subid=participants(proj)[1], task="balloonanalogrisktask")
  }

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})


# Get event files for a specific subject and task
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  x <- bids_project(ds001_path)
```

```
  files <- event_files(x, subid="01", task="balloonanalogrisktask")

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`file_pairs` *Find File Pairs in a BIDS Project*

Description

This function matches pairs of related files (e.g., BOLD and event files) in a BIDS project, returning a tibble with matched filenames. It's useful for verifying that corresponding files exist for each subject and task, such as ensuring every BOLD file has an associated events file.

Usage

```
file_pairs(
  x,
  pair = c("bold-events", "preproc-events"),
  task = ".*",
  matchon = c("run", "task"),
  ...
)
```

Arguments

`x`	A `bids_project` object.
`pair`	A character string specifying which pair of files to match. Currently supported: • "bold-events": matches BOLD files with event files • "preproc-events": matches preprocessed BOLD files with event files
`task`	A regex pattern to filter tasks. Default is ".*" (no filter).
`matchon`	A character vector of keys to match on, usually c("run", "task").
`...`	Additional arguments passed to internal functions.

Value

A tibble with columns:

- `subid`: The subject ID
- `task`: The task name
- `[type1]`: The name of the first file type (e.g., "bold" or "preproc")
- `[type2]`: The matched file of the second type (e.g., "events"), or `NA` if no match found
- Additional columns for matched metadata (e.g., run, session)

Examples

```
# Create a BIDS project object
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Match BOLD files with their corresponding event files
  bold_pairs <- file_pairs(proj, pair="bold-events")

  # Check pairs for a specific task
  task_pairs <- file_pairs(proj,
                           pair="bold-events",
                           task="balloonanalogrisktask")

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

flat_list *Get "flat" representation of BIDS Project*

Description

This function returns a flattened (non-hierarchical) representation of a BIDS project formatted as a data frame. It extracts file paths or file names from the BIDS tree structure, filtering for entries that start with "sub-" to focus on subject-level data.

Usage

```
flat_list(x, ...)

## S3 method for class 'bids_project'
flat_list(x, full_path = TRUE, ...)
```

Arguments

x	the bids_project object
...	extra args passed to methods
full_path	If TRUE, return full paths to files; if FALSE, return just file names (default: TRUE)

Value

A data frame containing either full paths to files (if full_path=TRUE) or just the file names (if full_path=FALSE). Each row represents one file in the BIDS project.

Examples

```
# Get flat representation with full paths
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  flat_list(proj)

  # Get flat representation with just file names
  flat_list(proj, full_path=FALSE)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`fmap_parser` *Fieldmap parser constructor*

Description

Fieldmap parser constructor

Usage

```
fmap_parser()
```

Value

A fieldmap BIDS parser object for parsing fieldmap files

Examples

```
# Create a fieldmap parser
parser <- fmap_parser()

# Parse a fieldmap file
result <- parse(parser, "sub-01_magnitude1.nii.gz")
```

fmriprep_anat_parser *fMRIPrep anatomical parser constructor*

Description

fMRIPrep anatomical parser constructor

Usage

```
fmriprep_anat_parser()
```

Value

An fMRIPrep anatomical parser object for parsing preprocessed anatomical files

Examples

```
# Create an fMRIPrep anatomical parser
parser <- fmriprep_anat_parser()

# Parse a preprocessed anatomical file
result <- parse(parser, "sub-01_space-MNI152NLin2009cAsym_desc-preproc_T1w.nii.gz")
```

fmriprep_func_parser *fMRIPrep functional parser constructor*

Description

fMRIPrep functional parser constructor

Usage

```
fmriprep_func_parser()
```

Value

An fMRIPrep functional parser object for parsing preprocessed functional files

Examples

```
# Create an fMRIPrep functional parser
parser <- fmriprep_func_parser()

# Parse a preprocessed functional file
result <- parse(parser, "sub-01_task-rest_space-MNI152NLin2009cAsym_desc-preproc_bold.nii.gz")
```

func_parser *Functional parser constructor*

Description

Functional parser constructor

Usage

```
func_parser()
```

Value

A functional BIDS parser object for parsing functional MRI files

Examples

```
# Create a functional parser
parser <- func_parser()

# Parse a functional file
result <- parse(parser, "sub-01_task-rest_run-01_bold.nii.gz")
```

func_scans *Get functional scans from a BIDS project*

Description

This function extracts functional scan files from a BIDS project that match specified criteria such as subject ID, task name, run number, and session. It can return either full paths or relative paths to the files.

Usage

```
func_scans(x, ...)

## S3 method for class 'mock_bids_project'
func_scans(
  x,
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  kind = "bold",
  suffix = "nii(\\.gz)?$",
  full_path = TRUE,
  ...
)
```

Arguments

x	A mock_bids_project object
...	Additional arguments passed to search_files
subid	Regex to match subject IDs (default: ".*")
task	Regex to match tasks (default: ".*")
run	Regex to match runs (default: ".*")
session	Regex to match sessions (default: ".*")
kind	Type of functional data (default: "bold")
suffix	Regex pattern for file suffix (default: "nii(\.gz)?$")
full_path	If TRUE, return full file paths (default: TRUE)

Value

A character vector of file paths to functional scans matching the criteria. Returns NULL if no matching files are found.

Examples

```
# Create a BIDS project object
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Get all functional scans
  all_scans <- func_scans(proj)

  # Get scans for specific subjects
  if (length(participants(proj)) > 0) {
    sub_scans <- func_scans(proj, subid=participants(proj)[1])
  }

  # Get scans for a specific task and run
  if (length(tasks(proj)) > 0) {
    task_scans <- func_scans(proj, task=tasks(proj)[1], run="01")
  }

  # Get scans with relative paths
  rel_scans <- func_scans(proj, full_path=FALSE)

  # Also try with a dataset that has sessions
  ds007_path <- get_example_bids_dataset("ds007")
  ds007_proj <- bids_project(ds007_path)
  if (length(sessions(ds007_proj)) > 0) {
    session_scans <- func_scans(ds007_proj, session=sessions(ds007_proj)[1])
  }

  # Clean up
  unlink(ds001_path, recursive=TRUE)
  unlink(ds007_path, recursive=TRUE)
```

```
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`func_scans.bids_project` *Get Functional Scans from a BIDS Project*

Description

This method extracts functional scan files from a BIDS project based on specified criteria such as subject ID, task name, run number, and session. It can return either full or relative file paths to the functional scans.

Usage

```
## S3 method for class 'bids_project'
func_scans(
  x,
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  kind = "bold",
  full_path = TRUE,
  ...
)
```

Arguments

`x`	A `bids_project` object.
`subid`	Regular expression for matching subject IDs. Default is ".*".
`task`	Regular expression for matching task names. Default is ".*".
`run`	Regular expression for matching run numbers. Default is ".*".
`session`	Regular expression for matching session IDs. Default is ".*".
`kind`	Regular expression for matching scan type. Default is "bold".
`full_path`	Logical. If TRUE, return full file paths. If FALSE, return relative paths. Default is TRUE.
`...`	Additional arguments (not currently used).

Value

A character vector of file paths to functional scans matching the criteria. Returns NULL if:

- No matching files are found
- The project doesn't contain functional data
- The specified criteria don't match any files

Examples

```
# Create a BIDS project
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Get all functional scans
  all_scans <- func_scans(proj)

  # Get scans for specific subjects
  sub_scans <- func_scans(proj, subid="0[123]")

  # Get scans for a specific task
  task_scans <- func_scans(proj, task="rest")

  # Get scans from specific runs
  run_scans <- func_scans(proj, run="0[123]")

  # Combine multiple filters
  filtered_scans <- func_scans(proj,
                               subid="01",
                               task="rest",
                               run="01")

  # Get relative paths instead of full paths
  rel_scans <- func_scans(proj, full_path=FALSE)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

get_example_bids_dataset

Download Example BIDS Dataset

Description

Downloads and extracts an example BIDS dataset for testing and demonstration purposes. The datasets are sourced from the official BIDS examples repository on GitHub.

Usage

```
get_example_bids_dataset(dataset_name = "ds001")
```

Arguments

dataset_name	Character string specifying which dataset to download. Common options include "ds001", "ds002", "ds007", "phoneme_stripped", etc.

Details

This function requires an internet connection to download data from GitHub. The datasets are cached in the temporary directory AND in memory for the session, so repeated calls with the same dataset_name will reuse the already downloaded data. Note: Don't call `unlink()` on the returned path in examples, as this defeats the caching mechanism and forces re-downloads.

Value

Character string containing the path to the downloaded dataset directory.

Examples

```
tryCatch({
  ds_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds_path)
  print(participants(proj))

  # Dataset cache is intentionally retained for performance.
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

get_repetition_time *Get Repetition Time (TR) from a sidecar JSON*

Description

This function attempts to find and return the repetition time (TR) for a given subject, task, and run (and optionally session) by locating the associated BOLD sidecar JSON file and extracting the 'RepetitionTime' field. If not found, returns NA.

Usage

```
get_repetition_time(x, subid, task, run = ".*", session = ".*", ...)
```

Arguments

x	A `bids_project` object.
subid	Subject ID (exact or regex).
task	Task name (exact or regex).
run	Run number (exact or regex). Default is ".*" to allow flexible matching.
session	Session ID (exact or regex). Default is ".*".
...	Additional arguments passed to `read_sidecar()`.

Value

A numeric value representing the RepetitionTime in seconds, or NA if not found.

Examples

```
# Download and get TR for a specific subject and task
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  if (length(participants(proj)) > 0 && length(tasks(proj)) > 0) {
    tr <- get_repetition_time(proj,
                              subid=participants(proj)[1],
                              task=tasks(proj)[1])
    cat("TR:", tr, "seconds\n")
  }

  # Try with a dataset that has sessions
  ds007_path <- get_example_bids_dataset("ds007")
  ds007_proj <- bids_project(ds007_path)
  if (length(participants(ds007_proj)) > 0 && length(sessions(ds007_proj)) > 0) {
    tr_session <- get_repetition_time(ds007_proj,
                                      subid=participants(ds007_proj)[1],
                                      session=sessions(ds007_proj)[1])
    cat("TR with session:", tr_session, "seconds\n")
  }

  # Clean up
  unlink(ds001_path, recursive=TRUE)
  unlink(ds007_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`infer_tr` *Infer TR (Repetition Time) from a BOLD file or sidecar*

Description

Given a path to a BOLD NIfTI file (`*.nii` or `*.nii.gz`) or its JSON sidecar (`*.json`), this function locates the appropriate sidecar JSON and returns the TR (in seconds). It prefers the JSON `RepetitionTime` field (BIDS-compliant). If that is not available, it falls back to computing TR as the median difference of `VolumeTiming` (if present). Optionally, when the sidecar cannot be found or is missing both fields, the function attempts to read TR from the NIfTI header (pixdim[4]) if an appropriate reader is installed.

Usage

```
infer_tr(x, ...)

## S3 method for class 'character'
infer_tr(
  x,
  prefer = c("json", "nifti"),
  fallback = TRUE,
  coerce_units = c("strict", "auto"),
  verbose = FALSE,
  ...
)
```

Arguments

x	A character path to a BOLD .nii[.gz] file or its .json sidecar, or a bids_project object.
...	Additional arguments passed to methods.
prefer	Preferred source of TR: "json" (default) or "nifti".
fallback	If TRUE (default), attempt NIfTI header fallback when JSON is not available or incomplete.
coerce_units	Unit handling for non-compliant values. "strict" (default) assumes seconds as per BIDS and returns values as-is. "auto" will convert clearly millisecond-like values to seconds (divide by 1000) and annotate the conversion in the return value's attributes.
verbose	If TRUE, print informative messages when falling back or when encountering special cases (e.g., SBRef files).

Details

For NIfTI inputs, the JSON sidecar is resolved by replacing the *.nii/*.nii.gz suffix with .json in the same directory. If that file is not found, the function searches the directory for a .json file with the same stem (filename without the NIfTI extension).

Value

Numeric TR in seconds, or NA_real_ if it cannot be determined. The return value includes attributes: source (e.g., json:RepetitionTime, json:VolumeTiming, nifti:pixdim4), path (the file used), and optionally variable = TRUE if VolumeTiming indicates non-constant TR; a unit = "ms->s" attribute is added if units were auto-converted.

Examples

```
tmp_json <- tempfile(fileext = ".json")
writeLines('{"RepetitionTime": 2}', tmp_json)
infer_tr(tmp_json)
unlink(tmp_json)
```

```
tmp_json2 <- tempfile(fileext = ".json")
writeLines('{"VolumeTiming": [0, 2, 4, 6]}', tmp_json2)
infer_tr(tmp_json2)
unlink(tmp_json2)
```

`list_confound_sets` *List available confound sets*

Description

Returns the names and short descriptions of the predefined confound sets usable with `confound_set()`.

Usage

```
list_confound_sets()
```

Value

A data.frame with columns `set` and `description`.

Examples

```
list_confound_sets()
```

`list_confound_strategies` *List available confound strategies*

Description

Returns the names and short descriptions of the predefined confound strategies usable with `confound_strategy()`.

Usage

```
list_confound_strategies()
```

Value

A data.frame with columns `strategy` and `description`.

Examples

```
list_confound_strategies()
```

`list_pack_bids` *List Contents of Packed BIDS Archive*

Description

This function lists the contents of a BIDS archive created by `pack_bids`, showing file sizes and identifying which files are stubs.

Usage

```
list_pack_bids(archive_path, verbose = TRUE)
```

Arguments

`archive_path`	Character string specifying the path to the archive file.
`verbose`	Logical. Whether to print summary statistics. Default is TRUE.

Value

A data frame with columns:

`file`	Relative file path within the archive
`size`	File size in bytes
`is_stub`	Logical indicating if the file is a 0-byte stub
`is_downsampled`	Logical indicating if the file is a downsampled image
`type`	File type based on extension (imaging, imaging_stub, imaging_downsampled, json, tsv, etc.)

Examples

```
# Create and inspect a packed BIDS archive
tryCatch({
  ds_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds_path)
  archive_path <- pack_bids(proj, verbose = FALSE)

  # List contents
  contents <- list_pack_bids(archive_path)

  # Show stub files
  stub_files <- contents[contents$is_stub, ]
  print(head(stub_files))

  # Clean up
  unlink(archive_path)
  unlink(ds_path, recursive = TRUE)
}, error = function(e) {
  message("Example failed: ", e$message)
```

```
})
```

`load_all_events` *Load All Event Files*

Description

Searches for and reads event files (`events.tsv`) from a BIDS project, combining them into a single (potentially nested) tibble.

This function searches for all `events.tsv` files that match the provided filters (subid, task, run, session) and loads them into a single tibble. If `full_path=TRUE`, full file paths are returned; otherwise relative paths.

Usage

```
load_all_events(x, ...)

## S3 method for class 'bids_project'
load_all_events(
  x,
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  full_path = TRUE,
  ...
)
```

Arguments

`x`	A `bids_project` object.
`...`	Additional arguments passed on to `search_files`.
`subid`	A regex for matching participant IDs. Default is `".*"`.
`task`	A regex for matching tasks. Default is `".*"`.
`run`	A regex for matching runs. Default is `".*"`.
`session`	A regex for matching sessions. Default is `".*"`.
`full_path`	If TRUE, return full file paths before reading. Default is TRUE.

Value

A tibble containing the combined event data.

A tibble combining all matched event files, with columns `.subid`, `.task`, `.run`, `.session` and all event columns. If no events are found, returns an empty tibble.

Examples

```
# Example with a bids_project (assuming events exist)
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  all_events <- load_all_events(proj)
  print(all_events)

  # Load specific subject/task
  if (length(participants(proj)) > 0) {
   sub01_events <- load_all_events(proj, subid=participants(proj)[1], task="balloonanalogrisktask")
    print(sub01_events)
  }

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

mask_files *Query Mask Files from a BIDS Project*

Description

Retrieves paths to brain mask files from a BIDS project, optionally filtered by subject, session, and coordinate space. Mask files are typically found in fMRIPrep derivatives and include brain masks and tissue segmentation masks.

Usage

```
mask_files(
  x,
  subid = ".*",
  session = ".*",
  space = ".*",
  full_path = TRUE,
  ...
)

## S3 method for class 'bids_project'
mask_files(
  x,
  subid = ".*",
  session = ".*",
  space = ".*",
  full_path = TRUE,
```

```
  ...
)

## S3 method for class 'mock_bids_project'
mask_files(
  x,
  subid = ".*",
  session = ".*",
  space = ".*",
  full_path = TRUE,
  ...
)
```

Arguments

x	A bids_project or mock_bids_project object.
subid	Regex pattern to match subject IDs (without "sub-" prefix). Default ".*" matches all subjects.
session	Regex pattern to match session IDs (without "ses-" prefix). Default ".*" matches all sessions.
space	Regex pattern to match coordinate space (e.g., "MNI152NLin2009cAsym", "T1w"). Default ".*" matches all spaces.
full_path	Logical. If TRUE (default), return absolute file paths. If FALSE, return paths relative to project root.
...	Additional arguments passed to search_files.

Value

Character vector of file paths matching the criteria, or NULL if no matching files are found.

Examples

```
# Get all mask files
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep = TRUE)

  # All masks
  all_masks <- mask_files(proj)

  # Masks in MNI space
  mni_masks <- mask_files(proj, space = "MNI152")

  # Masks for specific subject
  sub01_masks <- mask_files(proj, subid = "01")

  # Clean up
  unlink(ds_path, recursive = TRUE)
}, error = function(e) {
```

```
  message("Example requires internet connection: ", e$message)
})
```

---

pack_bids *Pack BIDS Project with Stub or Downsampled Imaging Files*

Description

This function creates a compressed archive (tar.gz or zip) of a BIDS project, either replacing large imaging files (.nii, .nii.gz) with 0-byte stub files or downsampling them to lower resolution while preserving all metadata files (JSON, TSV, etc.) with their full content. This is useful for sharing BIDS project structure and metadata without the large imaging data.

Usage

```
pack_bids(
  x,
  output_file = NULL,
  format = NULL,
  include_derivatives = TRUE,
  downsample_factor = NULL,
  downsample_method = "box",
  ncores = 1,
  max_file_size = "10MB",
  exclude = NULL,
  strict_bids = FALSE,
  verbose = TRUE,
  temp_dir = NULL,
  cleanup = TRUE
)
```

Arguments

x	A bids_project object created by bids_project.
output_file	Character string specifying the output archive filename. Should end with ".tar.gz" or ".zip". If not specified, defaults to "{project_name}_metadata.tar.gz" in the current directory.
format	Character string specifying the archive format. Can be "tar.gz" (default) or "zip". If not specified, inferred from output_file extension.
include_derivatives	Logical. Whether to include fMRIPrep derivatives if available. Default is TRUE.
downsample_factor	Numeric value between 0 and 1 specifying the downsampling factor for imaging files. If NULL (default), creates stub files. A value of 0.25 reduces dimensions by 4x (e.g., 64x64x64 becomes 16x16x16). Time dimension is preserved for 4D files.

`downsample_method`
: Character string specifying the downsampling method. Currently only "box" (box averaging) is supported. Default is "box".

`ncores`
: Integer specifying the number of cores for parallel processing during downsampling. Default is 1 (sequential). Values > 1 enable parallel processing if the 'future' package is available.

`max_file_size`
: Character string or numeric value specifying the maximum file size for non-imaging files to include. Files larger than this will be replaced with stub files. Can be specified as "1MB", "500KB", "1.5GB" or as numeric bytes. Default is "10MB".

`exclude`
: Character string with a regular expression pattern to exclude files. Files matching this pattern will be replaced with stub files. For example, "\.h5$" to exclude HDF5 files. Default is NULL (no exclusion).

`strict_bids`
: Logical. If TRUE, only include files that match BIDS naming conventions and standard BIDS metadata files. Non-BIDS files like .DS_Store, temporary files, or other non-standard files will be excluded. Default is FALSE (include all files).

`verbose`
: Logical. Whether to print progress messages. Default is TRUE.

`temp_dir`
: Character string specifying the temporary directory for creating the archive. If NULL (default), uses tempdir().

`cleanup`
: Logical. Whether to clean up the temporary directory after creating the archive. Default is TRUE.

Details

The function works by:

1. Creating a temporary copy of the BIDS project structure
2. Replacing all .nii and .nii.gz files with 0-byte stub files
3. Preserving all other files (JSON, TSV, TXT, etc.) with full content
4. Creating a compressed archive of the modified structure

This allows researchers to share BIDS dataset structure and metadata without the large imaging files, which is useful for:

- Sharing dataset organization and metadata for review
- Creating lightweight references for dataset structure
- Testing BIDS tools without full datasets

Value

Character string containing the path to the created archive file. Returns NULL if the operation fails.

Examples

```
# Create a BIDS project and pack it
tryCatch({
  ds_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds_path)

  # Pack with default settings (tar.gz with stub files)
  archive_path <- pack_bids(proj)

  # Pack with size limit and exclusion pattern
  archive_filtered <- pack_bids(proj,
                                max_file_size = "1MB",
                                exclude = "\\.h5$",
                                output_file = "ds001_filtered.tar.gz")

  # Pack with downsampling (4x reduction)
  archive_downsampled <- pack_bids(proj,
                                   downsample_factor = 0.25,
                                   output_file = "ds001_low4x.tar.gz")

  # Pack with downsampling using parallel processing
  if (requireNamespace("future", quietly = TRUE)) {
    archive_parallel <- pack_bids(proj,
                                  downsample_factor = 0.5,
                                  ncores = 2,
                                  output_file = "ds001_low2x.tar.gz")
  }

  # Pack as zip file
  zip_path <- pack_bids(proj, output_file = "ds001_metadata.zip")

  # Pack without derivatives
  archive_no_deriv <- pack_bids(proj, include_derivatives = FALSE)

  # Pack with strict BIDS mode (exclude non-BIDS files)
  archive_strict <- pack_bids(proj, strict_bids = TRUE,
                              output_file = "ds001_strict.tar.gz")

  # Clean up
  unlink(c(archive_path, archive_filtered, archive_downsampled, zip_path,
           archive_no_deriv, archive_strict))
  if (exists("archive_parallel")) unlink(archive_parallel)
  unlink(ds_path, recursive = TRUE)
}, error = function(e) {
  message("Example failed: ", e$message)
})
```

---

parse *Parse a file-name into BIDS components*

Description

This generic function parses a BIDS filename into its component parts. It uses a parser combinator approach to match the filename against known BIDS patterns and extract relevant metadata such as subject ID, session, task, run, and modality.

Usage

```
parse(x, fname, ...)
```

Arguments

x	the parser object to use for parsing
fname	the string (filename) to parse
...	extra args passed to methods

Value

A parsed representation of the BIDS filename, typically a list with extracted components

Examples

```
# Parse an anatomical file
parser <- anat_parser()
parse(parser, "sub-01_T1w.nii.gz")

# Parse a functional file
parser <- func_parser()
parse(parser, "sub-01_task-rest_run-01_bold.nii.gz")

# Use the generic BIDS parser
parser <- bids_parser()
parse(parser, "sub-01_ses-pre_task-rest_run-01_bold.nii.gz")
```

---

participants *Get participants from a BIDS project*

Description

This function retrieves a vector of unique participant IDs from a BIDS project. It extracts the subject identifiers from the project's data table, filtering out any NA values. Participant IDs in BIDS typically follow the format 'sub-XX'.

Usage

```
participants(x, ...)

## S3 method for class 'bids_project'
participants(x, ...)
```

Arguments

x	the bids_project object
...	extra args passed to methods

Value

A character vector of unique participant IDs found in the BIDS project. If no participants are found or the 'subid' column doesn't exist in the project's data table, returns an empty character vector.

Examples

```
# Get participants from a BIDS project
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  participants(proj)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

participants.mock_bids_project

Get Participants from Mock BIDS Project

Description

Extracts the unique participant IDs from the mock project definition. Note: Returns IDs *without* the "sub-" prefix for consistency with bids_project methods.

Usage

```
## S3 method for class 'mock_bids_project'
participants(x, ...)
```

Arguments

x	A mock_bids_project object.
...	Extra arguments (ignored).

Value

Character vector of unique participant IDs (e.g., c("01", "02")), sorted.

Examples

```
# Create a mock project
parts <- data.frame(participant_id = c("sub-01", "sub-02"))
fs <- data.frame(subid=c("01", "02"), datatype="func", suffix="bold.nii.gz", fmriprep=FALSE)
mock_proj <- create_mock_bids("SimpleMock", parts, fs)

# Get participant IDs
participants(mock_proj)
```

`plot.bids_confounds` *Plot PCA confounds*

Description

Visualize principal component scores and loadings returned by `read_confounds(..., npcs = ...)`. When multiple runs are present, the default view facets per run for scores (up to `max_panels`) and aggregates loadings across runs.

Usage

```
## S3 method for class 'bids_confounds'
plot(
  x,
  view = c("auto", "run", "aggregate"),
  pcs = NULL,
  top_n = 8,
  max_panels = 6,
  ...
)
```

Arguments

`x`	A `bids_confounds` object returned by `read_confounds()`.
`view`	Character. One of `"auto"`, `"run"`, or `"aggregate"`.
`pcs`	Integer or character vector of PCs to plot (e.g., `1:5` or `c("PC1", "PC2")`).
`top_n`	Integer. Keep the top `top_n` variables per PC based on absolute loading. Set to `NULL` to keep all variables.
`max_panels`	Integer. In `view = "auto"`, facet score plots only when the number of runs is at most `max_panels`.
`...`	Unused.

Value

A ggplot object, or a list of ggplot objects when patchwork is not available.

Examples

```
parts <- c("01", "02")
fs <- tibble::tibble(
  subid = rep(c("01", "02"), each = 2),
  datatype = "func",
  suffix = rep(c("bold.nii.gz", "desc-confounds_timeseries.tsv"), 2),
  task = "rest", fmriprep = TRUE
)
conf_data <- list()
for (p in parts) {
  key <- paste0("derivatives/fmriprep/sub-", p,
                "/func/sub-", p, "_task-rest_desc-confounds_timeseries.tsv")
  conf_data[[key]] <- data.frame(
    csf = rnorm(100), white_matter = rnorm(100),
    global_signal = rnorm(100), framewise_displacement = abs(rnorm(100)),
    trans_x = rnorm(100), trans_y = rnorm(100), trans_z = rnorm(100),
    rot_x = rnorm(100), rot_y = rnorm(100), rot_z = rnorm(100)
  )
}
mock <- create_mock_bids("ConfPlot", parts, fs, confound_data = conf_data)
conf <- read_confounds(mock, npcs = 3)
if (requireNamespace("ggplot2", quietly = TRUE)) {
  plot(conf)
}
```

---

plot.bids_project *Plot a BIDS project as a dendrogram*

Description

This method visualises the hierarchical file structure of a BIDS project. The tree is converted to a dendrogram and drawn using base graphics. Large projects can be trimmed by setting a maximum depth.

Usage

```
## S3 method for class 'bids_project'
plot(x, max_depth = Inf, ...)

## S3 method for class 'mock_bids_project'
plot(x, max_depth = Inf, ...)
```

Arguments

x	A bids_project object.
max_depth	Maximum depth of the tree to display. Defaults to Inf so the full hierarchy is shown.
...	Additional arguments passed to graphics::plot.

Value

The input object x is returned invisibly.

Examples

```
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  plot(proj)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

plot_bids *Plot a comprehensive visual overview of a BIDS project*

Description

This function creates a multi-panel visualization of a BIDS project structure, showing file distributions, completeness, and data characteristics.

Usage

```
plot_bids(
  x,
  interactive = TRUE,
  color_scheme = "viridis",
  include_derivatives = TRUE,
  file_size_scale = "log",
  highlight_missing = TRUE,
  visualization_mode = "standard",
  debug = FALSE
)
```

Arguments

x	A bids_project object
interactive	Logical. Whether to create an interactive plot (default TRUE)
color_scheme	Character. Name of the color palette to use (default "viridis")
include_derivatives	Logical. Whether to include derivatives data in the visualization (default TRUE)
file_size_scale	Character. Whether to scale file sizes ("log", "sqrt", or "linear", default "log")

highlight_missing
: Logical. Whether to highlight missing data points (default TRUE)

visualization_mode
: Character. The mode of visualization to use ("standard", "heatmap", or "complete")

debug
: Logical. Whether to print debugging information (default FALSE)

Value

A plot object (ggplot2, plotly, or other depending on settings)

Examples

```
# Create a basic BIDS project and plot it
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  plot_bids(proj)

  # Create an interactive plot
  plot_bids(proj, interactive=TRUE)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

preproc_scans *Get preprocessed functional MRI scans*

Description

This function retrieves paths to preprocessed functional MRI scans from a BIDS project. It searches for files in the fMRIPrep derivatives directory that match specified criteria, such as subject ID, task, run, and other BIDS metadata. Preprocessed scans are identified by having either 'desc-preproc' or 'kind-preproc' in their filename.

Usage

```
preproc_scans(
  x,
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  variant = NULL,
```

```
  space = ".*",
  modality = "bold",
  kind = ".*",
  full_path = FALSE,
  ...
)
```

Arguments

x	A bids_project object
subid	Subject ID regex to match specific subjects (default: ".*" for all subjects)
task	Task regex to match specific tasks (default: ".*" for all tasks)
run	Run regex to match specific runs (default: ".*" for all runs)
session	Session regex to match specific sessions (default: ".*" for all sessions)
variant	Preprocessing variant to match (default: NULL, which matches files without a variant)
space	Space regex to match specific spaces (default: ".*" for all spaces)
modality	Image modality to match (default: "bold" for functional MRI)
kind	Kind regex to match specific kinds (default: ".*" for all kinds)
full_path	If TRUE, return full file paths; if FALSE, return paths relative to the project root (default: FALSE)
...	Additional arguments passed to internal functions

Value

A character vector of file paths to preprocessed functional scans matching the criteria. If no matching files are found, returns NULL.

Examples

```
# Get all preprocessed scans from a BIDS project with fMRIPrep derivatives

# Download and load a BIDS project with fMRIPrep derivatives
tryCatch({
  ds001_deriv_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds001_deriv_path, fmriprep=TRUE)

  # Get all preprocessed scans
  scans <- preproc_scans(proj)

  # Get preprocessed scans for a specific subject
  if (!is.null(scans) && length(scans) > 0) {
    sub01_scans <- preproc_scans(proj, subid="01")
  }

  # Clean up
  unlink(ds001_deriv_path, recursive=TRUE)
```

```
}, error = function(e) {
  message("Example requires derivatives dataset: ", e$message)
})
```

`preproc_scans.bids_project` *Get preprocessed scans from a BIDS project*

Description

This function retrieves paths to preprocessed functional MRI scans from a BIDS project's fMRIPrep derivatives. It allows filtering by various BIDS entities such as subject, task, run, session, and space. The function is particularly useful for accessing preprocessed data for analysis pipelines.

Usage

```
## S3 method for class 'bids_project'
preproc_scans(
  x,
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  variant = NULL,
  space = ".*",
  modality = "bold",
  kind = ".*",
  full_path = FALSE,
  ...
)
```

Arguments

`x`	A `bids_project` object.
`subid`	A regex pattern for matching subjects. Default is ".*".
`task`	A regex pattern for matching tasks. Default is ".*".
`run`	A regex pattern for matching runs. Default is ".*".
`session`	A regex pattern for matching sessions. Default is ".*".
`variant`	A regex pattern for matching preprocessing variants. Default is NULL (no variant filtering).
`space`	A regex pattern for matching spaces (e.g., "MNI152NLin2009cAsym"). Default is ".*".
`modality`	A regex pattern for matching modality. Default is "bold". Set this to something else if you need a different modality.

kind	The kind of preprocessed data to return. Default is ".*" to match any kind.
full_path	If TRUE, return full file paths. Otherwise return relative paths. Default is FALSE.
...	Additional key-value filters for BIDS entities. These are matched against parsed file entities in the derivatives tree. Common examples: space = "MNI152NLin2009cAsym", res = "2", acq = "ap", echo = "1". Values are treated as regex. Keys already covered by explicit arguments (subid, task, run, session, space, variant, modality, kind) are ignored in ....

Value

A character vector of file paths to preprocessed scans matching the criteria. Returns NULL if:

- No matching files are found
- The project doesn't have fMRIPrep derivatives
- The specified criteria don't match any files

Examples

```
# Create a BIDS project with fMRIPrep derivatives
tryCatch({
  ds_path <- get_example_bids_dataset("phoneme_stripped")
  proj <- bids_project(ds_path, fmriprep=TRUE)

  # Get all preprocessed BOLD scans
  all_scans <- preproc_scans(proj)

  # Get preprocessed scans for specific subjects
  sub_scans <- preproc_scans(proj, subid="0[12]")

  # Get scans in MNI space
  mni_scans <- preproc_scans(proj, space="MNI152NLin2009cAsym")

  # Get scans for a specific task with full paths
  task_scans <- preproc_scans(proj,
                              task="phoneme",
                              full_path=TRUE)

  # Get scans from a specific session
  session_scans <- preproc_scans(proj, session="test")

  # Combine multiple filters
  filtered_scans <- preproc_scans(proj,
                                  subid="01",
                                  task="phoneme",
                                  run="01",
                                  space="MNI152NLin2009cAsym")

  # Filter by resolution (BIDS entity 'res')
  res2_scans <- preproc_scans(proj, res = "2")
```

```
  # Clean up
  unlink(ds_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`print.mock_bids_project` *Print Mock BIDS Project Summary*

Description

Provides a console summary of the mock BIDS project, displaying key information like participant count, tasks, sessions, derivatives status, and discovered BIDS entities.

Usage

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

Arguments

`x`	A `mock_bids_project` object.
`...`	Extra arguments (ignored).

Value

The `mock_bids_project` object `x` invisibly.

Examples

```
# Create a simple mock project
parts <- data.frame(participant_id = "01")
fs <- data.frame(subid = "01", datatype="func", suffix="bold.nii.gz", fmriprep=FALSE)
mock_proj <- create_mock_bids("SimpleMock", parts, fs)

# Print the summary
print(mock_proj)
```

`read_confounds` *Read Confound Files from a BIDS Project*

Description

This function reads in fMRIPrep confound tables for one or more subjects from a BIDS project. Confound files contain nuisance variables that can be used for denoising fMRI data, such as motion parameters, physiological signals, and other noise components. The function can optionally perform PCA reduction on the confounds and return either nested or flat tibbles.

Usage

```
read_confounds(x, ...)
```

Arguments

`x`	The object to read confounds from (typically a `bids_project`).
`...`	Additional arguments passed to methods, including: • `subid`: Regex to match subject IDs (default: ".*") • `task`: Regex to match tasks (default: ".*") • `session`: Regex to match sessions (default: ".*") • `run`: Regex to match runs (default: ".*") • `cvars`: Character vector of confound variable names to select • `npcs`: Integer. Perform PCA reduction and return this many PCs • `perc_var`: Numeric. Perform PCA reduction to retain this percentage of variance • `nest`: Logical. If TRUE, nests confound tables by subject/task/session/run (default: TRUE)

Value

A `bids_confounds` tibble containing confound data. If `nest=TRUE` (default), returns a nested tibble with columns for subject, task, session, run, and a nested `data` column containing the confound variables. If `nest=FALSE`, returns a flat tibble with all confound variables. When PCA is requested, the object includes a `pca` attribute with loadings/variance for plotting. Returns NULL if no matching files are found.

Examples

```
# Create a BIDS project with fMRIPrep derivatives
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep=TRUE)

  # Read all confound files
  all_conf <- read_confounds(proj)
```

```
  # Read confounds for specific subjects and tasks
  sub_conf <- read_confounds(proj,
                             subid="01",
                             task="balloonanalogrisktask")

  # Select specific confound variables
  motion_conf <- read_confounds(proj,
                                cvars=c("framewise_displacement",
                                        "trans_x", "trans_y", "trans_z",
                                        "rot_x", "rot_y", "rot_z"))

  # Perform PCA reduction
  pca_conf <- read_confounds(proj, npcs=5)

  # Get confounds as a flat tibble
  flat_conf <- read_confounds(proj, nest=FALSE)

  # Combine multiple options
  custom_conf <- read_confounds(proj,
                                subid="01",
                                task="balloonanalogrisktask",
                                cvars=c("framewise_displacement",
                                        "trans_x", "trans_y", "trans_z"),
                                npcs=3,
                                nest=FALSE)

  # Clean up
  unlink(ds_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`read_confounds.bids_project` *Read confound files*

Description

Reads in fmriprep confound tables for one or more subjects.

Usage

```
## S3 method for class 'bids_project'
read_confounds(
  x,
  subid = ".*",
  task = ".*",
  session = ".*",
```

```
  run = ".*",
  cvars = DEFAULT_CVARS,
  npcs = -1,
  perc_var = -1,
  nest = TRUE,
  ...
)
```

Arguments

x	A bids_project object
subid	Subject ID regex
task	Task regex
session	Session regex
run	Run regex. If the run identifier cannot be extracted from the filename, the run value defaults to "1".
cvars	The names of the confound variables to select. Defaults to DEFAULT_CVARS. Canonical names such as "csf" are automatically mapped to any matching column names found in the dataset using CVARS_ALIASES. You can also pass convenience sets from confound_set(), e.g., confound_set("motion24"), or wildcard patterns like "cosine_*", "motion_outlier_*", or "a_comp_cor_*[6]".
npcs	Perform PCA reduction on confounds and return npcs PCs.
perc_var	Perform PCA reduction to retain perc_var% variance.
nest	If TRUE, nests confound tables by subject/task/session/run.
...	Additional arguments (not currently used)

Value

A bids_confounds tibble (nested if nest=TRUE) with identifier columns for participant_id, task, session, and run. When PCA is requested, the object includes a pca attribute with per-run loadings and variance used by plot().

Examples

```
# Try to load a BIDS project with fMRIPrep derivatives
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep=TRUE)

  # Read confounds with canonical names (automatically resolve to actual columns)
  conf <- read_confounds(proj, cvars = c("csf", "framewise_displacement"))

  # Use convenience sets
  conf_36p <- read_confounds(proj, cvars = confound_set("36p"))
  conf_compcor6 <- read_confounds(proj, cvars = confound_set("acompcor", n = 6))

  # Read confounds for specific subjects and tasks
  conf_sub <- read_confounds(proj, subid="01", task="balloonanalogrisktask")
```

```
  # Get confounds as flat tibble
  conf_flat <- read_confounds(proj, nest=FALSE)

  # Clean up
  unlink(ds_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires derivatives dataset with confounds: ", e$message)
})
```

`read_confounds.mock_bids_project` *Read Confound Files (Mock Implementation)*

Description

Read Confound Files (Mock Implementation)

Usage

```
## S3 method for class 'mock_bids_project'
read_confounds(
  x,
  subid = ".*",
  task = ".*",
  session = ".*",
  run = ".*",
  cvars = NULL,
  npcs = -1,
  perc_var = -1,
  nest = TRUE,
  ...
)
```

Arguments

`x`	A `mock_bids_project` object.
`subid`	Regex pattern for subject IDs. Default `".*"`.
`task`	Regex pattern for task names. Default `".*"`.
`session`	Regex pattern for session IDs. Default `".*"`.
`run`	Regex pattern for run indices. Default `".*"`.
`cvars`	Variables to select (ignored in mock).
`npcs`	PCA components (applied when requested).
`perc_var`	PCA variance (applied when requested).
`nest`	If `TRUE`, returns a nested tibble keyed by subject, task, session and run.
`...`	Additional BIDS entities (passed to `search_files`).

Value

A bids_confounds tibble of confound data (nested if nest = TRUE).

Examples

```
parts <- c("01")
fs <- tibble::tibble(
  subid = "01", datatype = "func",
  suffix = c("bold.nii.gz", "desc-confounds_timeseries.tsv"),
  task = "rest", fmriprep = c(TRUE, TRUE)
)
conf_data <- list()
key <- "derivatives/fmriprep/sub-01/func/sub-01_task-rest_desc-confounds_timeseries.tsv"
conf_data[[key]] <- data.frame(
  csf = rnorm(50), white_matter = rnorm(50),
  trans_x = rnorm(50), trans_y = rnorm(50)
)
mock <- create_mock_bids("ConfTest", parts, fs, confound_data = conf_data)
conf <- read_confounds(mock)
print(conf)
```

---

read_events *Read Event Files from a BIDS Project*

Description

This generic function reads and nests event files from a BIDS project. Event files contain timing information about task events, conditions, and responses during functional MRI scans. The function can filter events by subject and task, and returns a nested tibble for easy data manipulation.

Usage

```
read_events(x, ...)
```

Arguments

x	The object to read events from (typically a bids_project).
...	Additional arguments passed to methods.

Value

A nested tibble with columns:

- .task: Task name
- .run: Run number
- .subid: Subject ID
- data: Nested column containing the event data If no matching data is found, returns an empty tibble with appropriate columns.

Examples

```
# Create a BIDS project
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Read all event files
  all_events <- read_events(proj)

  # Read events for specific subjects
  sub_events <- read_events(proj, subid="0[123]")

  # Read events for a specific task
  task_events <- read_events(proj, task="balloonanalogrisktask")

  # Combine multiple filters
  filtered_events <- read_events(proj,
                                 subid="01",
                                 task="balloonanalogrisktask")

  # Access nested data
  if (nrow(filtered_events) > 0) {
    first_run <- filtered_events$data[[1]]
    print(head(first_run))
  }

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`read_events.bids_project`

Read event files from a BIDS project

Description

Reads and nests event files for given subjects and tasks from a `bids_project` object. Returns a nested tibble with event data grouped by task, session, run, and subject. Event files typically contain trial-by-trial information for task-based fMRI data, including onset times, durations, trial types, and other task-specific variables.

Usage

```
## S3 method for class 'bids_project'
read_events(x, subid = ".*", task = ".*", run = ".*", session = ".*", ...)
```

Arguments

x	A bids_project object.
subid	Regex pattern to match subject IDs. Default is ".*" (all subjects).
task	Regex pattern to match tasks. Default is ".*" (all tasks).
run	Regex pattern to match runs. Default is ".*" (all runs).
session	Regex pattern to match sessions. Default is ".*" (all sessions).
...	Additional arguments passed to event_files.

Value

A nested tibble with columns:

- .task: Task name
- .session: Session ID (if present)
- .run: Run number
- .subid: Subject ID
- data: A nested tibble containing the event data with columns:
  - onset: Event onset time in seconds
  - duration: Event duration in seconds
  - Additional task-specific columns (e.g., trial type, response, accuracy) If no matching data is found, returns an empty tibble with appropriate columns. Run and session identifiers are parsed from filenames using func_parser().

Examples

```
# Create a BIDS project object
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)

  # Read all event files
  all_events <- read_events(proj)

  # Read events for a specific subject and task
  sub01_events <- read_events(proj,
                              subid="01",
                              task="balloonanalogrisktask")

  # Read events for multiple subjects and a specific run
  multi_sub_events <- read_events(proj,
                                  subid="0[1-3]",
                                  run="01")

  # Access nested data for analysis
  if (nrow(sub01_events) > 0) {
    # Get first subject's data
    first_sub_data <- sub01_events$data[[1]]
```

```
    # Calculate mean trial duration
    mean_duration <- mean(first_sub_data$duration)
  }

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`read_events.mock_bids_project` *Read Event Files from Mock BIDS Project*

Description

Retrieves and formats event data stored within the mock project object.

Usage

```
## S3 method for class 'mock_bids_project'
read_events(x, subid = ".*", task = ".*", run = ".*", session = ".*", ...)
```

Arguments

`x`	A `mock_bids_project` object.
`subid`	Regex pattern for subject IDs. Default `".*"`.
`task`	Regex pattern for task names. Default `".*"`.
`run`	Regex pattern for run indices. Default `".*"`.
`session`	Regex pattern for session IDs. Default `".*"`.
`...`	Additional arguments passed to `event_files`.

Value

A nested tibble with columns `.subid`, `.task`, `.run`, `.session` (if applicable), and `data` (containing the event tibbles), or an empty tibble if no matching data.

Examples

```
parts <- c("01")
fs <- tibble::tibble(
  subid = "01", datatype = "func",
  suffix = c("bold.nii.gz", "events.tsv"),
  task = "rest", run = "01", fmriprep = FALSE
)
```

```
evt_data <- list()
evt_data[["sub-01/func/sub-01_task-rest_run-01_events.tsv"]] <-
  tibble::tibble(onset = c(1, 5, 10), duration = c(0.5, 0.5, 0.5),
                 trial_type = c("go", "stop", "go"))
mock <- create_mock_bids("EventTest", parts, fs, event_data = evt_data)
events <- read_events(mock)
print(events)
```

`read_func_scans.bids_project` *Read in a set of four-dimensional functional scans*

Description

Read in a set of four-dimensional functional scans

Usage

```
read_func_scans.bids_project(
  x,
  mask,
  mode = c("normal", "bigvec"),
  subid = "^sub-.*",
  task = ".*",
  run = ".*",
  modality = "bold",
  ...
)
```

Arguments

`x`	A `bids_project` object
`mask`	A brain mask of type `LogicalNeuroVol`
`mode`	The file mode: 'normal' for in-memory files or 'bigvec' for on-disk files
`subid`	One or more subject IDs (regex)
`task`	An optional task regex
`run`	An optional run regex
`modality`	The image modality (usually "bold")
`...`	Extra arguments passed to `neuroim2::read_vec`

Value

An instance of type `NeuroVec`

Examples

```
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep=TRUE)
  mask <- brain_mask(proj, subid="01")
  vec <- read_func_scans.bids_project(proj, mask,
                                      subid="01",
                                      task="balloonanalogrisktask",
                                      run="01")
  unlink(ds_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires derivatives dataset: ", e$message)
})
```

`read_preproc_scans.bids_project` *Read preprocessed functional MRI scans from a BIDS project*

Description

This function reads preprocessed functional MRI scans from a BIDS project's fMRIPrep derivatives directory. It uses the `preproc_scans` function to locate the files and then reads them into a `NeuroVec` object using the neuroim2 package. If a mask is not provided, one will be automatically created from available brainmask files.

Usage

```
read_preproc_scans.bids_project(
  x,
  mask = NULL,
  mode = c("normal", "bigvec"),
  subid = "^sub-.*",
  task = ".*",
  run = ".*",
  modality = "bold",
  ...
)
```

Arguments

`x`	A `bids_project` object with fMRIPrep derivatives
`mask`	A brain mask of type `LogicalNeuroVol`, or NULL (if NULL, a mask will be created automatically)
`mode`	The file mode: 'normal' for in-memory files or 'bigvec' for on-disk files
`subid`	Regular expression to match subject IDs (default: "^sub-.*" to match all subjects)

task	Regular expression to match tasks (default: ".*" to match all tasks)
run	Regular expression to match runs (default: ".*" to match all runs)
modality	Image modality to match (default: "bold" for functional MRI)
...	Extra arguments passed to neuroim2::read_vec

Details

This function requires the neuroim2 package to be installed. It will throw an error if the package is not available or if fMRIPrep derivatives are not found in the BIDS project. If no mask is provided, it will create one using the create_preproc_mask function.

Value

An instance of type NeuroVec containing the preprocessed functional data.

Examples

```
# Load a BIDS project with fMRIPrep derivatives
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep=TRUE)

  # Read preprocessed scans for all subjects
  # (mask will be created automatically)
  all_scans <- read_preproc_scans(proj)

  # Read preprocessed scans for a specific subject
  sub01_scans <- read_preproc_scans(proj, subid="01")

  # Read preprocessed scans for a specific task and run
  task_scans <- read_preproc_scans(proj,
                                   task="balloonanalogrisktask",
                                   run="01")

  # Specify mode for large datasets
  bigvec_scans <- read_preproc_scans(proj, mode="bigvec")

  # Provide a custom mask
  mask <- create_preproc_mask(proj, thresh=0.95)
  masked_scans <- read_preproc_scans(proj, mask=mask)

  # Clean up
  unlink(ds_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires derivatives dataset: ", e$message)
})
```

`read_sidecar` *Read sidecar JSON files and return metadata as a tidy tibble*

Description

This function searches for JSON sidecar files matching the given criteria (subject, task, run, session), reads the JSON content, and converts all top-level fields into columns of a tibble. Each file's metadata becomes one row in the returned tibble. This is particularly useful for extracting metadata about BIDS imaging files, such as acquisition parameters, task descriptions, and other relevant information.

Usage

```
read_sidecar(
  x,
  subid = ".*",
  task = ".*",
  run = ".*",
  session = ".*",
  modality = "bold",
  full_path = TRUE,
  ...
)
```

Arguments

`x`	A `bids_project` object.
`subid`	A regex for matching subject IDs. Default is `".*"`.
`task`	A regex for matching tasks. Default is `".*"`.
`run`	A regex for matching runs. Default is `".*"`.
`session`	A regex for matching sessions. Default is `".*"`.
`modality`	A regex for matching modality/kind (e.g. "bold"). Default is `"bold"`. This is matched against the 'kind' field in parsed BIDS filenames.
`full_path`	If TRUE, return full file paths in the `file` column. Default is TRUE.
`...`	Additional arguments passed to `search_files()`.

Value

A tibble with one row per JSON file. Columns include:

- `file`: the JSON file path
- `.subid`: subject ID extracted from filename
- `.session`: session ID extracted from filename (if present)
- `.task`: task name extracted from filename (if present)

- .run: run number extracted from filename (if present)
- Additional columns for each top-level key in the JSON files If no files are found, returns an empty tibble.

Examples

```
# Read all BOLD sidecar files from a BIDS dataset
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  metadata <- read_sidecar(proj)

  # Read sidecar files for a specific subject and task
  sub01_meta <- read_sidecar(proj,
                             subid="01",
                             task="balloonanalogrisktask")

  # Read sidecar files for anatomical data
  anat_meta <- read_sidecar(proj,
                            modality="T1w",
                            full_path=FALSE)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

---

search_files *Search files in BIDS structure*

Description

This function searches for files in a BIDS project that match a specified pattern and optional key-value criteria. It can be used to find files in both raw data and preprocessed derivatives based on filename patterns and BIDS metadata.

This function searches for files in a BIDS project that match a specified pattern and optional key-value criteria. It can search in both raw data and preprocessed derivatives (if available).

Finds files in the mock BIDS tree by matching file names and BIDS entities.

Usage

```
search_files(x, ...)

## S3 method for class 'bids_project'
search_files(x, regex = ".*", full_path = FALSE, strict = TRUE, ...)
```

```
## S3 method for class 'mock_bids_project'
search_files(x, regex = ".*", full_path = FALSE, strict = TRUE, ...)
```

Arguments

x	A mock_bids_project object.
...	Additional BIDS entities to match (e.g., subid = "01", task = "rest"). Values are treated as regex patterns unless they are simple strings without regex characters.
regex	A regular expression to match filenames (node names). Default ".*".
full_path	If TRUE, return full paths (prefixed with x$path). If FALSE, return relative paths within the BIDS structure. Default FALSE.
strict	If TRUE (default), queries for a BIDS entity (e.g., task="X") require the entity to exist on the file node and match the pattern. If FALSE, files lacking the queried entity are not automatically excluded (though they won't match if the pattern isn't .*).

Value

A character vector of file paths matching the criteria, or NULL if no matches found.

A character vector of file paths matching the criteria, or NULL if no matches found.

A character vector of matching file paths, or NULL if no matches.

Examples

```
# Search for event files in a BIDS dataset
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path, fmriprep=FALSE)
  event_files <- search_files(proj, regex="events\\.tsv$")

  # Search with additional criteria
  sub01_files <- search_files(proj, regex="bold\\.nii\\.gz$", subid="01",
                              task="balloonanalogrisktask")

  # Get full paths
  full_paths <- search_files(proj, regex="events\\.tsv$", full_path=TRUE)

  # Search with strict matching
  strict_matches <- search_files(proj, regex="\\.tsv$", strict=TRUE,
                                 task="balloonanalogrisktask")

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

```
# Search for event files in a BIDS dataset
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path, fmriprep=FALSE)
  event_files <- search_files(proj, regex="events\\.tsv$")

  # Search with additional criteria (note: ds001 only has one subject '01')
  sub01_files <- search_files(proj, regex="bold\\.nii\\.gz$", subid="01",
                              task="balloonanalogrisktask")

  # Get full paths
  full_paths <- search_files(proj, regex="events\\.tsv$", full_path=TRUE)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`sessions` *Get sessions from a BIDS project*

Description

This function retrieves a vector of session IDs from a BIDS project. Sessions in BIDS are typically represented as directories named 'ses-XX' within subject directories. This function extracts and returns the unique session identifiers.

Usage

```
sessions(x, ...)

## S3 method for class 'bids_project'
sessions(x, ...)
```

Arguments

`x`	the object to extract sessions from
`...`	extra args passed to methods

Value

A character vector of unique session IDs if the project has sessions, or NULL if the project does not have sessions

Examples

```
# Get sessions from a BIDS project
tryCatch({
  ds007_path <- get_example_bids_dataset("ds007")
  proj <- bids_project(ds007_path)
  sessions(proj)

  # Dataset cache is intentionally retained for performance.
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`sessions.mock_bids_project` *Get Sessions from Mock BIDS Project*

Description

Extracts the unique session IDs found in the mock project's file structure. Note: Returns IDs *without* the "ses-" prefix.

Usage

```
## S3 method for class 'mock_bids_project'
sessions(x, ...)
```

Arguments

`x`	A `mock_bids_project` object.
`...`	Extra arguments (ignored).

Value

Character vector of unique session IDs (e.g., c("pre", "post")), sorted, or `NULL` if the project does not have sessions.

Examples

```
# Create a mock project with sessions
parts <- data.frame(participant_id = "01")
fs <- data.frame(subid="01", session="test", datatype="func", suffix="bold.nii.gz", fmriprep=FALSE)
mock_proj <- create_mock_bids("SessionMock", parts, fs)

# Get session IDs
sessions(mock_proj)

# Project without sessions
```

```
fs_no_session <- data.frame(subid="01", datatype="func", suffix="bold.nii.gz", fmriprep=FALSE)
mock_proj_no_sess <- create_mock_bids("NoSessionMock", parts, fs_no_session)
sessions(mock_proj_no_sess) # Returns NULL
```

---

surface_files *Query Surface Files from a BIDS Project*

Description

Retrieves paths to surface mesh files (GIFTI format, .gii) from a BIDS project, optionally filtered by hemisphere and surface type. Surface files are typically found in fMRIPrep derivatives and represent cortical surface reconstructions in various coordinate spaces.

Usage

```
surface_files(
  x,
  subid = ".*",
  session = ".*",
  hemi = ".*",
  surf_type = ".*",
  space = ".*",
  full_path = TRUE,
  ...
)

## S3 method for class 'bids_project'
surface_files(
  x,
  subid = ".*",
  session = ".*",
  hemi = ".*",
  surf_type = ".*",
  space = ".*",
  full_path = TRUE,
  ...
)

## S3 method for class 'mock_bids_project'
surface_files(
  x,
  subid = ".*",
  session = ".*",
  hemi = ".*",
  surf_type = ".*",
  space = ".*",
  full_path = TRUE,
```

```
  ...
)
```

Arguments

x	A bids_project or mock_bids_project object.
subid	Regex pattern to match subject IDs (without "sub-" prefix). Default ".*" matches all subjects.
session	Regex pattern to match session IDs (without "ses-" prefix). Default ".*" matches all sessions.
hemi	Hemisphere filter: "L" for left, "R" for right, or ".*" for both. Default ".*" matches both hemispheres.
surf_type	Surface type filter: "pial", "inflated", "midthickness", "smoothwm", "white", "sphere", "spherereg", or ".*" for all types. Default ".*" matches all surface types.
space	Regex pattern to match coordinate space (e.g., "fsnative", "fsaverage"). Default ".*" matches all spaces.
full_path	Logical. If TRUE (default), return absolute file paths. If FALSE, return paths relative to project root.
...	Additional arguments passed to search_files.

Value

Character vector of file paths matching the criteria, or NULL if no matching files are found.

Examples

```
# Get all surface files
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep = TRUE)

  # All surfaces
  all_surfs <- surface_files(proj)

  # Left hemisphere pial surfaces only
  left_pial <- surface_files(proj, hemi = "L", surf_type = "pial")

  # All surfaces in fsnative space
  fsnative_surfs <- surface_files(proj, space = "fsnative")

  # Clean up
  unlink(ds_path, recursive = TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

`tasks` *Get tasks from a BIDS project*

Description

This function retrieves a sorted vector of unique task names from a BIDS project. Tasks in BIDS are typically represented in filenames with the pattern 'task-XX'. This function extracts and returns the unique task identifiers, filtering out any NULL or NA values.

Usage

```
tasks(x, ...)

## S3 method for class 'bids_project'
tasks(x, ...)
```

Arguments

`x`	the object to extract tasks from
`...`	extra args passed to methods

Value

A character vector of unique, sorted task names found in the BIDS project

Examples

```
# Get tasks from a BIDS project
tryCatch({
  ds001_path <- get_example_bids_dataset("ds001")
  proj <- bids_project(ds001_path)
  tasks(proj)

  # Clean up
  unlink(ds001_path, recursive=TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

tasks.mock_bids_project *Get Tasks from Mock BIDS Project*

Description

Extracts the unique task names found in the mock project's file structure. Note: Returns names *without* the "task-" prefix.

Usage

```
## S3 method for class 'mock_bids_project'
tasks(x, ...)
```

Arguments

x	A mock_bids_project object.
...	Extra arguments (ignored).

Value

Character vector of unique task names (e.g., c("rest", "nback")), sorted.

Examples

```
# Create a mock project with tasks
parts <- data.frame(participant_id = "01")
fs <- data.frame(subid="01", task="taskA", run="01", datatype="func",
                 suffix="bold.nii.gz", fmriprep=FALSE)
fs <- rbind(fs, data.frame(subid="01", task="taskB", run="01", datatype="func",
                           suffix="bold.nii.gz", fmriprep=FALSE))
mock_proj <- create_mock_bids("TaskMock", parts, fs)

# Get task names
tasks(mock_proj)
```

transform_files *Query Transform Files from a BIDS Project*

Description

Retrieves paths to transformation files (xfm, warp, affine) from a BIDS project, optionally filtered by source and target coordinate space. Transform files are typically found in fMRIPrep derivatives and encode spatial transformations between different coordinate spaces (e.g., T1w to MNI, boldref to T1w).

Usage

```
transform_files(
  x,
  subid = ".*",
  session = ".*",
  from = ".*",
  to = ".*",
  mode = ".*",
  kind = ".*",
  full_path = TRUE,
  ...
)

## S3 method for class 'bids_project'
transform_files(
  x,
  subid = ".*",
  session = ".*",
  from = ".*",
  to = ".*",
  mode = ".*",
  kind = ".*",
  full_path = TRUE,
  ...
)

## S3 method for class 'mock_bids_project'
transform_files(
  x,
  subid = ".*",
  session = ".*",
  from = ".*",
  to = ".*",
  mode = ".*",
  kind = ".*",
  full_path = TRUE,
  ...
)
```

Arguments

x	A bids_project or mock_bids_project object.
subid	Regex pattern to match subject IDs (without "sub-" prefix). Default ".*" matches all subjects.
session	Regex pattern to match session IDs (without "ses-" prefix). Default ".*" matches all sessions.
from	Regex pattern to match source space (the "from" BIDS entity). Common values: "T1w", "boldref", "fsnative". Default ".*" matches all.

to	Regex pattern to match target space (the "to" BIDS entity). Common values: "MNI152NLin2009cAsym", "T1w", "fsnative". Default ".*" matches all.
mode	Regex pattern to match transform mode entity. Default ".*".
kind	Transform type: "xfm", "warp", "affine", or ".*" for all types. Default ".*" matches all transform types.
full_path	Logical. If TRUE (default), return absolute file paths. If FALSE, return paths relative to project root.
...	Additional arguments passed to search_files.

Value

Character vector of file paths matching the criteria, or NULL if no matching files are found.

Examples

```
# Get all transform files
tryCatch({
  ds_path <- get_example_bids_dataset("ds000001-fmriprep")
  proj <- bids_project(ds_path, fmriprep = TRUE)

  # All transforms
  all_xfms <- transform_files(proj)

  # Transforms from T1w to MNI space
  t1_to_mni <- transform_files(proj, from = "T1w", to = "MNI152")

  # All transforms for a specific subject
  sub01_xfms <- transform_files(proj, subid = "01")

  # Clean up
  unlink(ds_path, recursive = TRUE)
}, error = function(e) {
  message("Example requires internet connection: ", e$message)
})
```

Index