bids-matlab

Release v0.2.0

BIDS-MATLAB developers

May 01, 2024

CONTENT

1 BIDS for MATLAB / Octave **3**
- 1.1 Installation, Features . . . 3
- 1.2 Usage . . . 3
  - 1.2.1 Reading and writing JSON files . . . 3
- 1.3 Implementation . . . 4

2 Indexing and querying a BIDS dataset **5**
- 2.1 Example . . . 6
- 2.2 Examples . . . 8

3 BIDS file handling **9**
- 3.1 Examples . . . 9
  - 3.1.1 Example . . . 12
  - 3.1.2 Example . . . 13

4 Function description **17**
- 4.1 Example . . . 19
- 4.2 Example . . . 21

5 Utility functions **23**
- 5.1 Example . . . 23
- 5.2 Example . . . 27

6 BIDS stats model handling **31**
- 6.1 Examples . . . 31
  - 6.1.1 Example . . . 32
  - 6.1.2 Example . . . 34

7 Variable transformations **37**
- 7.1 Applying transformations . . . 37
  - 7.1.1 Example . . . 38
- 7.2 Basic operations . . . 38
- 7.3 Logical operations . . . 39
- 7.4 Munge operations . . . 39
  - 7.4.1 Assign . . . 39
  - 7.4.2 Concatenate . . . 41
  - 7.4.3 Copy . . . 41
  - 7.4.4 Delete . . . 41
  - 7.4.5 DropNA . . . 41
  - 7.4.6 Factor . . . 42
  - 7.4.7 Filter . . . 42

7.4.8 Label identical rows . . . 42
7.4.9 Merge identical rows . . . 43
7.4.10 Replace . . . 43
7.4.11 Select . . . 44
7.4.12 Split . . . 44
7.5 Compute operations . . . 44
7.5.1 Constant . . . 45
7.5.2 Mean . . . 45
7.5.3 Product . . . 46
7.5.4 Scale . . . 46
7.5.5 Std . . . 47
7.5.6 Sum . . . 47
7.5.7 Threshold . . . 47

8 Using the BIDS schema **49**
8.1 Example . . . 50
8.2 Example . . . 50
8.3 Example . . . 51
8.4 Example . . . 51
8.5 Example . . . 52
8.6 Example . . . 52
8.7 Example . . . 53
8.8 Example . . . 53
8.9 Example . . . 53

9 Performance **55**

10 developer documentation **57**
10.1 +internal . . . 57
10.1.1 Example . . . 58
10.1.2 Example . . . 62
10.1.3 Example . . . 62

11 Changelog **65**
11.1 [Unreleased] . . . 65
11.1.1 Added . . . 65
11.1.2 Changed . . . 65
11.1.3 Deprecated . . . 65
11.1.4 Removed . . . 65
11.1.5 Fixed . . . 65
11.1.6 Security . . . 66
11.2 [v0.2.0] . . . 66
11.2.1 Changed . . . 66
11.2.2 Deprecated . . . 66
11.2.3 Removed . . . 66
11.2.4 Fixed . . . 66
11.2.5 Security . . . 66
11.3 [v0.1.0] . . . 66
11.3.1 Changed . . . 66
11.3.2 Deprecated . . . 66
11.3.3 Removed . . . 66
11.3.4 Fixed . . . 66
11.3.5 Security . . . 66

12 Indices and tables **67**

MATLAB Module Index 69

Index 71

This repository aims at centralising MATLAB/Octave tools to interact with datasets conforming to the BIDS (Brain Imaging Data Structure) format.

For more information about BIDS, visit the BIDS website.

To see how to install BIDS-Matlab, please check the github repository.

CHAPTER

ONE

BIDS FOR MATLAB / OCTAVE

This repository aims at centralising MATLAB/Octave tools to interact with datasets conforming to the BIDS (Brain Imaging Data Structure) format.

For more information about BIDS, visit https://bids.neuroimaging.io/.

Join our chat on the BIDS-MATLAB channel on the brainhack mattermost and our google group.

See also PyBIDS for Python and the BIDS Starter Kit.

1.1 Installation, Features

Please see the relevant sections of the README

1.2 Usage

BIDS matlab is structured as package, so you can easily access functions in subfolders that start with +.

To use the `+bids/layout.m` function:

```
BIDS = bids.layout('/home/data/ds000117');
bids.query(BIDS, 'subjects')
```

To use the `+bids/+util/jsondecode.m` function:

```
content = bids.util.jsondecode('/home/data/some_json_file.json');
```

A tutorials are available as a Jupyter Notebooks and scripts and can be run interactively via Binder.

1.2.1 Reading and writing JSON files

If you are using MATLAB R2016b or newer, nothing else needs to be installed.

If you are using MATLAB R2016a or older, or using Octave, you need to install a supported JSON library for your MATLAB or Octave. This can be any of:

- JSONio for MATLAB or Octave
- SPM12

1.3 Implementation

Starting point was `spm_BIDS.m` from SPM12 (documentation) reformatted in a `+bids` package with dependencies to other SPM functions removed.

CHAPTER

TWO

INDEXING AND QUERYING A BIDS DATASET

BIDS-Matlab allows you to index a raw or derivative BIDS dataset with *bids.layout()*, and then query the content of that dataset with *bids.query()*.

The general API of these functions is detailed below.

For an example on how to use them, check out this jupyter notebook.

`bids.layout(`*varargin*`)`

Parse a directory structure formatted according to the BIDS standard

USAGE:

```
BIDS = bids.layout(pwd, ...
                   'use_schema', true, ...
                   'index_derivatives', false, ...
                   'index_dependencies', true, ...
                   'filter', struct([]), ...
                   'tolerant', true, ...
                   'verbose', false)
```

Parameters

- **root** (char) – directory of the dataset formatted according to BIDS [default: pwd]
- **use_schema** (logical) – If set to true, the parsing of the dataset will follow the bids-schema provided with bids-matlab. If set to false files just have to be of the form sub-label_[entity-label]_suffix.ext to be parsed.
- **index_derivatives** (logical) – if true this will index the content of the any derivatives folder in the BIDS dataset.
- **index_dependencies** (logical) – if true this will index the explicit dependencies (with "IntendedFor" in json files)
- **filter** (struct with optional fields sub, ses, modality. Regular expression can be used for sub and ses.) – Can be used to index only a subset of the dataset.
- **tolerant** (logical) – Set to true to turn validation errors into warnings
- **verbose** (logical) – Set to true to get more feedback

2.1 Example

```
BIDS = bids.layout(fullfile(get_test_data_dir(), '7t_trt'), ...
                   'use_schema', true, ...
                   'verbose', true, ...
                   'index_derivatives', false, ...
                   'filter', struct('sub', {{'^0[12]'}}, ...
                                    'modality', {{'anat', 'func'}}, ...
                                    'ses', {{'1', '2'}}));
```

bids.**query**(*BIDS*, *query*, *varargin*)

Queries a directory structure formatted according to the BIDS standard

USAGE:

```
result = bids.query(BIDS, query, filter)
```

Parameters

- **BIDS** (structure or char) – BIDS directory name or BIDS structure (from bids.layout)
- **query** (char) – type of query (see list below)

Type of queries allowed.

Any of the following:

- 'modalities': known as datatype in BIDS (anat, func, eeg...)
- 'entities'
- 'suffixes'
- 'data': filenames
- 'metadata': associated metadata (using the inheriance principle)
- 'metafiles': json sidecar files
- 'dependencies': associated files (for example the event.tsv for a bold.nii or eeg.eeg file)
- 'participants': content and metadata of participants.tsv
- 'phenotype': content and metadata of the phenotype folder
- 'extensions'
- 'tsv_content'

And any of the BIDS entities:

- 'acquisitions'
- 'atlases'
- 'ceagents'
- 'chunks'
- 'densities'
- 'descriptions'
- 'directions'

- 'echos'
- 'flips'
- 'hemispheres'
- 'inversions'
- 'labels'
- 'mtransfers'
- 'parts'
- 'processings'
- 'reconstructions'
- 'recordings'
- 'resolutions'
- 'sessions'
- 'subjects'
- 'runs'
- 'samples'
- 'spaces'
- 'splits'
- 'stains'
- 'tasks'
- 'tracers'

Warning: Note that most of the query types are plurals.

Parameters
filter (structure or nX2 cell or series of key-value parameters) – filter for the query

The choice of available keys to filter with includes:

- 'suffix'
- 'extension'
- 'prefix'
- 'modality'

It can also include any of the entity keys present in the files in the dataset. To know what those are, use:

```
bids.query(BIDS, 'entities')
```

Warning: Note that integers as query label for the entity keys listed below:

- 'run'

- 'flip'
- 'inv'
- 'split'
- 'echo'
- 'chunk'

If you want to exclude an entity, use `''` or `[]`.

It is possible to use regular expressions in the queried values.

2.2 Examples

Querying for 'BOLD' files for subject '01', for run 1 to 5 of the 'stopsignalwithpseudowordnaming' task with gunzipped nifti files.

```
data = bids.query(BIDS, 'data', ...
                        'sub', '01', ...
                        'task', 'stopsignalwithpseudowordnaming', ...
                        'run', 1:5, ...
                        'extension', '.nii.gz', ...
                        'suffix', 'bold');
```

Same as above but using a filter structure.

```
filters = struct('sub', '01', ...
                 'task', 'stopsignalwithpseudowordnaming', ...
                 'run', 1:5, ...
                 'extension', '.nii.gz', ...
                 'suffix', 'bold');

data = bids.query(BIDS, 'data', filters);
```

Same as above but using regular expression to query for subjects 1 to 5.

```
filters = {'sub', '0[1-5]'; ...
           'task', 'stopsignal.*'; ...
           'run', 1:5; ...
           'extension', '.nii.*'; ...
           'suffix', 'bold'};

data = bids.query(BIDS, 'data', filters);
```

The following query would return all files that do not contain the task entity.

```
data = bids.query(BIDS, 'data', 'task', '')
```

CHAPTER

THREE

BIDS FILE HANDLING

class bids.**File**

Class to deal with BIDS filenames

USAGE:

```
bf = bids.File(input, ...
                'use_schema', false, ...
                'tolerant', true,
                'verbose', false);
```

Parameters

- **input** (filename or structure) – path to the file or a structure with the file information
- **use_schema** (logical) – will apply the BIDS schema when parsing or creating filenames
- **tolerant** (logical) – turns errors into warning when set to true
- **verbose** (logical) – silences warnings

3.1 Examples

Initialize with a filename.

```
input = fullfile(pwd, 'sub-01_ses-02_T1w.nii');
bf = bids.File(input);
```

Initialize with a structure

```
input = struct('ext', '.nii', ...
               'suffix', 'T1w', ...
               'entities', struct('sub', '01', ...
                                  'ses', '02'));
bf = bids.File(input);
```

Remove prefixes and add a desc-preproc entity-label pair.

```
input = 'wuasub-01_ses-test_task-faceRecognition_run-02_bold.nii';
bf = bids.File(input, 'use_schema', false);
bf.prefix = '';
bf.entities.desc = 'preproc';
disp(file.filename)
```

Use the BIDS schema to get entities in the right order.

```
input.suffix = 'bold';
input.ext = '.nii';
input.entities = struct('sub', '01', ...
                        'acq', '1pt5', ...
                        'run', '02', ...
                        'task', 'face recognition');

bf = bids.File(name_spec, 'use_schema', true);
```

Load metadata (supporting inheritance).

```
bf = bids.File('tests/data/synthetic/sub-01/anat/sub-01_T1w.nii.gz');
```

Access metadata

```
bf.metadata()

  struct with fields:
    Manufacturer: 'Siemens'
    FlipAngle: 10
```

Modify metadata

```
% Adding new value

bf = bf.metadata_add('NewField', 'new value');
bf.metadata()

  struct with fields:
    manufacturer: 'siemens'
    flipangle: 10
    NewField: 'new value'

% Appending to existing value

bf = bf.metadata_append('NewField', 'new value 1');
bf.metadata()

  struct with fields:
    manufacturer: 'siemens'
    flipangle: 10
    NewField: {'new value'; 'new value 1'}

% Removing value

bf = bf.metadata_remove('NewField');
bf.metadata()

  struct with fields:
    manufacturer: 'siemens'
    flipangle: 10
```

Modify several fields of metadata

```
bf = bf.metadata_update('Description', 'source file', ...
                        'NewField', 'new value', ...
                        'manufacturer', []);
bf.metadata()

  struct with fields:
    flipangle: 10
    description: 'source file'
    NewField: 'new value'
```

Export metadata as json:

```
bf.metadata_write()
```

Property Summary

prefix
bids prefix

extension
file extension

suffix
file suffix

entities
list of entities

modality
name of file modality

path
absolute path

bids_path
path within dataset

filename
bidsified name

json_filename
bidsified name for json file

metadata_files
list of metadata files related

metadata
list of metadata for this file

entity_required
Required entities

entity_order
Expected order of entities

schema

BIDS schema used

Method Summary

update()

executed automatically before getting a value

reorder_entities(*entity_order*)

USAGE:

```
file = file.reorder_entities(entity_order);
```

Parameters

entity_order (`cell of char`) – Optional. The order of the entities.

If the no entity order is provided, it will try to rely on the schema to find an appropriate order

3.1.1 Example

```
% filename with ses entity in the wrong position
filename = 'wuasub-01_task-faceRecognition_ses-test_run-02_bold.nii';
file = bids.File(filename, 'use_schema', false);
file = file.reorder_entities({'sub', 'ses'});

% use the schema to do the reordering
filename = 'wuasub-01_task-faceRecognition_ses-test_run-02_bold.nii';
file = bids.File(filename, 'use_schema', false);
file = file.use_schema();
file = file.reorder_entities();
```

rename(*varargin*)

Renames a file according following some specification

USAGE:

```
file = file.rename('spec', spec, 'dry_run', true, 'verbose', [], 'force',␣
↪false);
```

Parameters

- **spec** (`structure`) – structure specifying what entities, suffix, extension… to apply If one of the entities in the `spec` contains a '.' it will be replaced by *pt*.
- **dry_run** (`logical`) – If `true` no file is actually renamed. `true` is the default to avoid renaming files by mistake.
- **verbose** (`logical`) – displays `input --> output`
- **force** (`logical`) – Overwrites existing file.

3.1.2 Example

```
%% rename an SPM preprocessed file

% expected_name = fullfile(pwd, ...
%                          'sub-01', ...
%                          'sub-01_task-faceRep_space-individual_desc-preproc_
↪bold.nii');

input_filename = 'uasub-01_task-faceRep_bold.nii';

file = bids.File(input_filename, 'use_schema', false);

spec.prefix = ''; % remove prefix
spec.entities.desc = 'preproc'; % add description entity
spec.entity_order = {'sub', 'task', 'desc'};

file = file.rename('spec', spec, 'dry_run', false, 'verbose', true);

%% Get a specific file from a dataset to rename

BIDS = bids.layout(path_to_dataset)

% construct a filter to get only the file we want/
subject = '001';
run = '001';
suffix = 'bold';
task = 'faceRep';
filter = struct('sub', subject, 'task', task, 'run', run, 'suffix', suffix);

file_to_rename = bids.query(BIDS, 'data', filter);

file = bids.File(file_to_rename, 'use_schema', false);

% specification to remove run entity
spec.entities.run = '';

% first run with dry_run = true to make sure we will get the expected output
file = file.rename('spec', spec, 'dry_run', true, 'verbose', true);

% rename the file by setting dry_run to false
file = file.rename('spec', spec, 'dry_run', false, 'verbose', true);
```

normalize_entities(~, *entities*, *replace*)

Clean up entities

Replaces "." in entity label with "pt".

USAGE:

```
entities = file.normalize_entities(entities);
```

use_schema()

Loads BIDS schema into instance and tries to update properties:

- file.modality
- file.required_entity
- file.entity_order
- file.relative_pth

USAGE:

```
file = file.use_schema();
```

validate_entities()

use entity_order got from schema as a proxy for allowed entity keys

USAGE:

```
file.validate_entities();
```

get_required_entities()

USAGE:

```
[file, required_entities] = file.get_required_entities()
```

get_modality_from_schema()

USAGE:

```
[file, modality] = file.get_modality_from_schema()
```

get_entity_order_from_schema()

USAGE:

```
[file, entity_order] = file.get_entity_order_from_schema()
```

check_required_entities()

USAGE:

```
file.check_required_entities()
```

metadata_update(*varargin*)

Update stored metadata with new values passed in varargin, which can be either a structure, or pairs of key-values.

See also

bids.util.update_struct

USAGE:

```
f = f.metadata_update(key1, value1, key2, value2);
f = f.metadata_update(struct(key1, value1, ...
                      key2, value2));
```

metadata_add(*field*, *value*)

Add a new field (or replace existing) to the metadata structure

metadata_append(*field*, *value*)

Append new value to a metadata.(field) If metadata.(field) is a chararray, it will be first transformed into cellarray.

metadata_remove(*field*)

Removes field from metadata

metadata_write(*varargin*)

Export current content of metadata to sidecar json with same name as current file.

Metadata fields can be modified with new values passed in varargin, which can be either a structure, or pairs of key-values. These modifications do not affect current File object, and only exported into file. Use bids.File.metadata_update to update current metadata. Returns full path to the exported sidecar json file.

See also

bids.util.update_struct

USAGE:

```
f.metadata_write(key1, value1, key2, value2);
f.metadata_write(struct(key1, value1, ...
                        key2, value2));
```

get_modality(*entities*)

Retrieves modality out of the path

Only works if ses and sub entities match those found in the path

CHAPTER

FOUR

FUNCTION DESCRIPTION

bids.**derivatives_json**(*varargin*)

Creates dummy content for a given BIDS derivative file.

USAGE:

```
json = derivatives_json(derivative_filename, 'force', false)
```

Parameters

- **derivative_filename** (char)
- **force** (logical) – when *true* it will force the creation of a json content even when the filename contains no BIDS derivatives entity.

bids.**init**(*varargin*)

Initialize dataset with README, description, folder structure...

USAGE:

```
bids.init(pth, ...
          'folders', folders, ,...
          'is_derivative', false,...
          'is_datalad_ds', false, ...
          'tolerant', true, ...
          'verbose', false)
```

Parameters

- **pth** (char) – directory where to create the dataset
- **folders** (structure) – define the folder structure to create. folders.subjects folders.sessions folders.modalities
- **is_derivative** (logical)
- **is_datalad_ds**

class bids.**Description**

Class to deal with dataset_description files.

USAGE:

```
ds_desc = bids.Description(pipeline, BIDS);
```

Parameters

- **pipeline** (char) – pipeline name
- **BIDS** (structure or char) – output from BIDS layout to identify the source dataset used when creating a derivatives dataset Can also be the path to a dataset_descriptin.json file

Constructor Summary

Description(*varargin*)

USAGE:

```
ds_desc = bids.Description(pipeline, BIDS);
```

Property Summary

content

dataset description content

is_derivative

logical

pipeline

name of the pipeline used to generate this derivative dataset

Method Summary

set_derivative()

USAGE:

```
ds_desc = ds_desc.set_derivative();
```

set_field(*varargin*)

USAGE:

```
ds_desc = ds_desc.set_field(key, value);
ds_desc = ds_desc.set_field(struct(key1, value1, ...
                                   key2, value2));
```

append(*key*, *value*)

Appends an item to the dataset description content.

USAGE:

```
ds_desc = ds_desc.append(key, value);
```

write(*folder*)

Writes json file of the dataset description.

USAGE:

```
ds_desc.write([folder = pwd]);
```

bids.**copy_to_derivative**(*varargin*)

Copy selected data from BIDS layout to given derivatives folder.

USAGE:

```
bids.copy_to_derivative(BIDS, ...
                        'pipeline_name', '', ...
                        'out_path', '', ...
                        'filters', struct(), ...
                        'unzip', true, ...
                        'force', false, ...
                        'skip_dep', false, ...
                        'use_schema, true, ...
                        'verbose', false, ...
                        'tolerant', false);
```

Parameters

- **BIDS** (`structure or char`) – BIDS directory name or BIDS structure (from bids.layout)
- **pipeline_name** (`char`) – name of pipeline to use
- **out_path** (`char`) – path to directory containing the derivatives
- **filter** (`structure or cell`) – list of filters to choose what files to copy (see bids.query)
- **unzip** (`logical`) – If `true` then all `.gz` files will be unzipped after being copied. For MacOS and Unix system, this will require a version of gunzip >= 1.6.
- **force** (`logical`) – If set to `false` it will not overwrite any file already present in the destination.
- **skip_dep** (`logical`) – If set to `false` it will copy all the dependencies of each file.
- **tolerant** (`boolean`) – Defaults to `false`. Set to `true` to turn errors into warnings.
- **use_schema** (`logical`) – If set to `true` it will only copy files that are BIDS valid.
- **verbose** (`logical`)

All the metadata of each file is read through the whole hierarchy and dumped into one side-car json file for each file copied. In practice this "unravels" the inheritance principle.

4.1 Example

```
dataset = fullfile(pwd, 'bids-examples', 'qmri_vfa');

output_path = fullfile(pwd, 'output');

filter =  struct('modality', 'anat',
                 'sub', '01');

pipeline_name = 'SPM12';

bids.copy_to_derivative(dataset, ...
                        'pipeline_name', pipeline_name, ...
                        'out_path', output_path, ...
                        'filter', filter, ...
                        'force', true, ...
                        'unzip', false, ...
                        'verbose', true);
```

bids.**report**(*varargin*)

Create a short summary of the acquisition parameters of a BIDS dataset.

The output can be saved to a markdown file and/or printed to the screen.

USAGE:

```
bids.report(BIDS,
            'filter', filter, ...
            'output_path', output_path, ...
            'read_nifti', read_nifti, ...
            'verbose', verbose);
```

Parameters

- **BIDS** (`char or structure`) – Path to BIDS dataset or output of bids.layout [Default = pwd]
- **filter** (`structure`) – Specifies which the subject, session, ... to take as template. [Default = struct('sub', '', 'ses', '')]. See bids.query for more information.
- **output_path** (`char`) – Folder where the report should be printed. If empty (default) then the output is sent to the prompt.
- **read_nifti** (`logical`) – If set to `true` (default) the function will try to read the NIfTI file to get more information. This relies on the `spm_vol.m` function from SPM.
- **verbose** (`logical`) – If set to `false` (default) the function does not output anything to the prompt.

bids.**validate**(*root*, *options*)

BIDS Validator

USAGE:

```
[sts, msg] = bids.validate(root, options)
```

Parameters

root – directory formatted according to BIDS [Default: pwd]

Returns

- **sts**
  `0` if successful
- **msg**
  warning and error messages

Command line version of the BIDS-Validator: https://github.com/bids-standard/bids-validator

Web version: https://bids-standard.github.io/bids-validator/

bids.**diagnostic**(*varargin*)

Create a diagnostic figure for a dataset.

- list the number of files for each subject split by: - modality - task (optional)
- list the number of trials for each event type

USAGE:

```
diagnostic_table = diagnostic(BIDS, ...
                              'use_schema', true, ...
                              'output_path', '', ...
                              'filter', struct(), ...
                              'split_by', {''})
```

Parameters

- **BIDS** (`structure or char`) – BIDS directory name or BIDS structure (from `bids.layout`)
- **split_by** (`cell`) – splits results by a given BIDS entity (now only `task` is supported)
- **use_schema** (`logical`) – If set to `true`, the parsing of the dataset will follow the bids-schema provided with bids-matlab. If set to `false` files just have to be of the form `sub-label_[entity-label]_suffix.ext` to be parsed. If a folder path is provided, then the schema contained in that folder will be used for parsing.
- **out_path** (`string`) – path to directory containing the derivatives
- **filter** (`structure or cell`) – list of filters to choose what files to copy (see bids.query)
- **trial_type_col** (`char`) – Optional. Name of the column containing the trial type. Defaults to `'trial_type'`.
- **verbose** (`logical`) – Optional. Set to false to not show the figure. Defaults to `true`.

4.2 Example

```
BIDS = bids.layout(path_to_dataset);
diagnostic_table = bids.diagnostic(BIDS, 'output_path', pwd);
diagnostic_table = bids.diagnostic(BIDS, 'split_by', {'task'}, 'output_path', pwd);
```

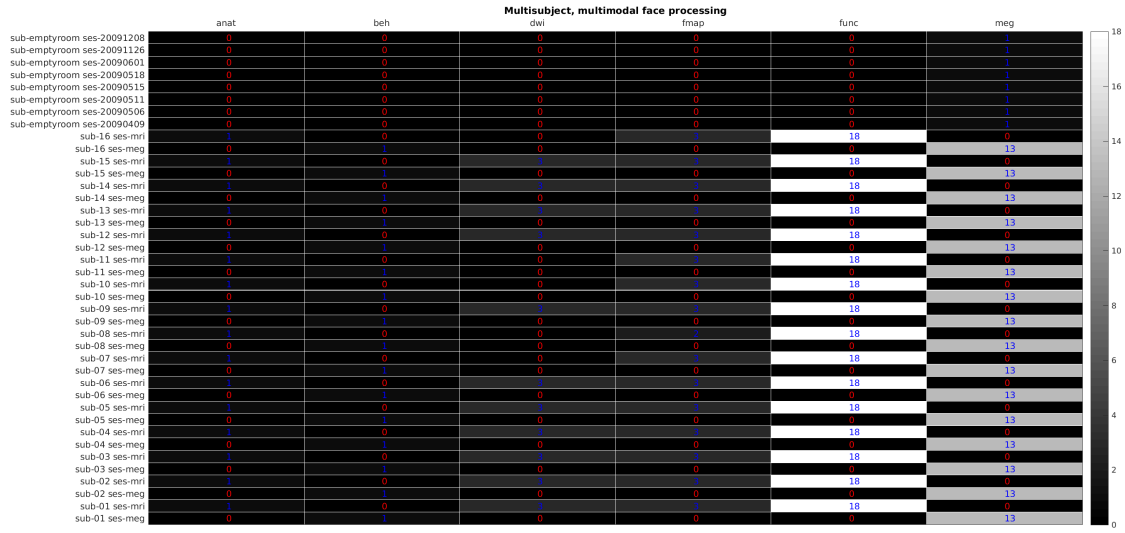

Fig. 1: output of `diagnostic`

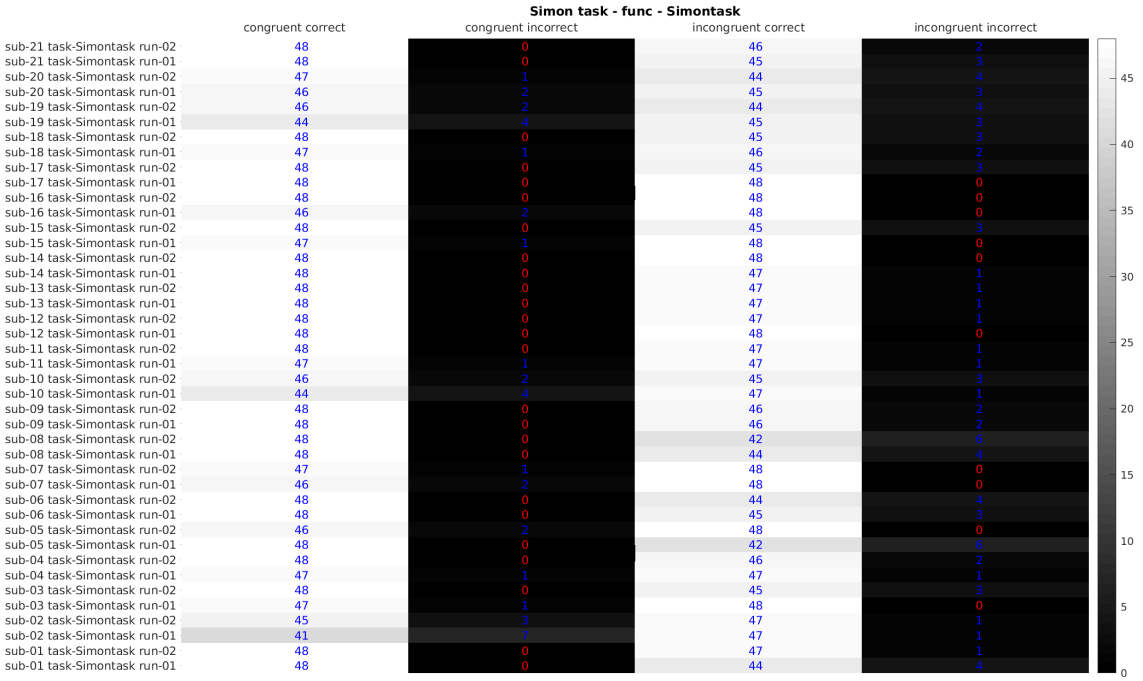

Fig. 2: output of `diagnostic` for events

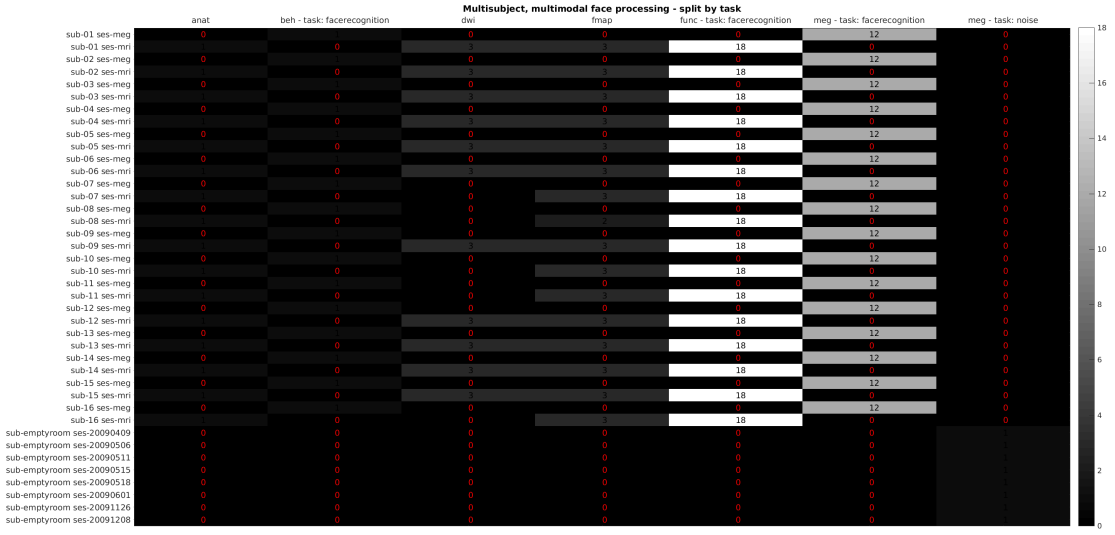

Fig. 3: output of `diagnostic` split by task

CHAPTER

FIVE

UTILITY FUNCTIONS

bids.util.**create_data_dict**(*varargin*)

Create a JSON data dictionary for a TSV file.

Levels in columns that may lead to invalid matlab structure fieldnames are renamed. Hence the output may need manual cleaning.

Descriptions may be added for columns if a match is found in the BIDS schema: for example: trial_type, onset...

To create better data dictionaries, please see the tools for hierarchical event descriptions.

USAGE:

```
data_dict = bids.util.create_data_dict(tsv_file, ...
                                       'output', [], ...
                                       'schema', true, ...
                                       'force', false, ...
                                       'level_limit', 10, ...
                                       'verbose', true);
```

Parameters

- **output** – filename for the output files. Can pass be a cell char of paths
- **force** (`logical`) – If set to `false` it will not overwrite any file already present in the destination.
- **schema** (`logical or a schema object`) – If set to `true` it will use the schema to try to find definitions for the column headers
- **level_limit** – Maximum number of levels to list. Defaults to 10;

5.1 Example

```
BIDS = bids.layout(pth_bids_example, 'ds001'));

tsv_files = bids.query(BIDS, 'data', ...
                       'sub', '01', ...
                       'suffix', 'events');

data_dict = bids.util.create_data_dict(tsv_files{1}, ...
                                       'output', 'tmp.json', ...
                                       'schema', true);
```

bids.util.**create_participants_tsv**(*varargin*)

Creates a simple participants tsv for a BIDS dataset.

USAGE:

```
output_filename = bids.util.create_participants_tsv(layout_or_path, ...
                                                    'use_schema', true, ...
                                                    'tolerant', true, ...
                                                    'verbose', false)
```

Parameters

- **layout_or_path** (path or structure)
- **use_schema** (logical)
- **tolerant** (logical) – Set to true to turn validation errors into warnings
- **verbose** (logical) – Set to true to get more feedback

bids.util.**create_readme**(*varargin*)

Create a README in a BIDS dataset.

USAGE:

```
bids.util.create_readme(layout_or_path, is_datalad_ds, ...
                        'tolerant', true, ...
                        'verbose', false)
```

Parameters

- **layout_or_path** (path or structure)
- **tolerant** (logical) – Set to true to turn validation errors into warnings
- **verbose** (logical) – Set to true to get more feedback

bids.util.**create_scans_tsv**(*varargin*)

Create a simple scans.tsv for each participant of a BIDS dataset.

USAGE:

```
output_filename = bids.util.create_scans_tsv(layout_or_path, ...
                                             'use_schema', true, ...
                                             'tolerant', true, ...
                                             'verbose', false)
```

Parameters

- **layout_or_path** (path or structure)
- **use_schema** (logical)
- **tolerant** (logical) – Set to true to turn validation errors into warnings
- **verbose** (logical) – Set to true to get more feedback

bids.util.**create_sessions_tsv**(*varargin*)

Create a simple sessions.tsv for each participant of a BIDS dataset.

USAGE:

```
output_filename = bids.util.create_sessions_tsv(layout_or_path, ...
                                                'use_schema', true, ...
                                                'tolerant', true, ...
                                                'verbose', false)
```

Parameters

- **layout_or_path** (path or structure)
- **use_schema** (logical)
- **tolerant** (logical) – Set to true to turn validation errors into warnings
- **verbose** (logical) – Set to true to get more feedback

bids.util.**download_ds**(*varargin*)

Downloads a BIDS data for a demo from a given source

USAGE:

```
output_dir = download_moae_ds('source', 'spm', ...
                              'demo', 'moae', ...
                              'out_path', fullfile(bids.internal.root_dir(), 'demos
↪'), ...
                              'force', false, ...
                              'verbose', true, ...
                              'delete_previous', true);
```

SPM:

```
bids.util.download_ds('source', 'spm', 'demo', 'moae')
bids.util.download_ds('source', 'spm', 'demo', 'facerep')
bids.util.download_ds('source', 'spm', 'demo', 'eeg')
```

—

BRAINSTORM:

```
bids.util.download_ds('source', 'brainstorm', 'demo', 'ieeg')
bids.util.download_ds('source', 'brainstorm', 'demo', 'meg')
```

ieeg: SEEG+MRI: 190 Mb

https://neuroimage.usc.edu/brainstorm/Tutorials/Epileptogenicity ftp://neuroimage.usc.edu/pub/tutorials/tutorial_epimap_bids.zip

meg: MEG+MRI+DWI: 208 Mb

https://neuroimage.usc.edu/brainstorm/Tutorials/FemMedianNerve ftp://neuroimage.usc.edu/pub/tutorials/sample_fem.zip

ecog: SEEG+ECOG+MRI: 897 Mb

https://neuroimage.usc.edu/brainstorm/Tutorials/ECoG ftp://neuroimage.usc.edu/pub/tutorials/sample_ecog.zip

meg_rest: MEG resting-state: 5.2 Gb

https://neuroimage.usc.edu/brainstorm/Tutorials/RestingOmega ftp://neuroimage.usc.edu/pub/tutorials/sample_omega.zip

bids.util.**jsondecode**(*file*, *varargin*)

Decode JSON-formatted file

USAGE:

```
json = bids.util.jsondecode(file, opts)
```

Parameters

- **file** (char) – name of a JSON file or JSON char
- **opts** (structure) – structure of optional parameters (only with JSONio):

`opt.replacementStyle`: char to control how non-alphanumeric characters are replaced.

- `'underscore'` Default
- `'hex'`
- `'delete'`
- `'nop'`

Returns

- **json**
  JSON structure

bids.util.**jsonencode**(*varargin*)

Encode data to JSON-formatted file

USAGE:

```
bids.util.jsonencode(filename, json, opts)
```

Parameters

- **filename** (char) – JSON filename
- **json** (structure) – JSON structure

USAGE:

```
S = bids.util.jsonencode(json, opts)
```

Parameters

- **json** (structure) – JSON structure
- **opts** – optional parameters

Returns

- **S**
  (char) serialized JSON structure

bids.util.**mkdir**(*varargin*)

Make new directory trees.

USAGE:

```
sts = bids.util.mkdir(dir, ...)
```

Parameters

dir (character array, or cell array of strings) – directory structure to create

Returns

- **sts**
  status is true if all directories were successfully created or already existing, false otherwise.

5.2 Example

```
bids.util.mkdir('dataset', {'sub-01', 'sub-02'}, {'mri', 'eeg'});
```

Based on spm_mkdir from SPM12

bids.util.**plot_events**(*varargin*)

USAGE:

```
plot_events(events_files, 'include', include, ...
                          'trial_type_col', 'trial_type', ...
                          'model_file', path_to_model)
```

Parameters

- **events_files** (path or cellstr of paths) – BIDS events TSV files.
- **include** (char or cellstr) – Optional. Restrict conditions to plot.
- **trial_type_col** (char or cellstr) – Optional. Defines the column where trial types are listed. Defaults to 'trial_type'
- **model_file** (fullpath) – Optional. Bids stats model file to apply to events.tsv before plotting

Example:

```
data_dir = fullfile(get_test_data_dir(), 'ds108');

BIDS = bids.layout(data_dir);

events_files = bids.query(BIDS, ...
                          'data', ...
                          'sub', '01', ...
                          'run', '01', ...
                          'suffix', 'events');

include = {'Reapp_Neg_Cue', 'Look_Neg_Cue', 'Look_Neutral_Cue'};
bids.util.plot_events(events_files, 'include', include);
```

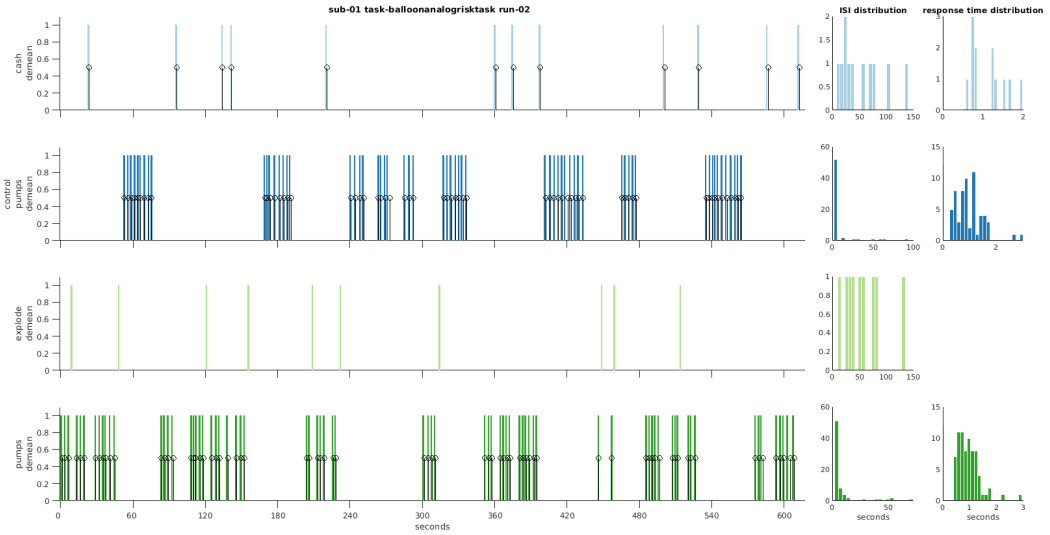

Fig. 1: output of `plot_events`

`bids.util.`**`tsvread`**(*filename*, *field_to_return*, *hdr*)

Load text and numeric data from tab-separated-value or other file.

USAGE:

```
file_content = tsvread(filename, field_to_return, hdr)
```

Parameters

- **`filename`** (`char`) – filename (can be gzipped) {txt,mat,csv,tsv,json}
- **`field_to_return`** – name of field to return if data stored in a structure [default: `''`]; or index of column if data stored as an array
- **`hdr`** (`logical`) – detect the presence of a header row for csv/tsv [default: `true`]

Returns

- **file_content**
  corresponding data array or structure

Based on spm_load.m from SPM12.

`bids.util.`**`tsvwrite`**(*filename*, *var*)

Save text and numeric data to tab-separated-value file

USAGE:

```
tsvwrite(filename, var)
```

Parameters

- **`filename`** (`string`)
- **`var`** (`data array or structure`)

Based on spm_save.m from SPM12.

CHAPTER

SIX

BIDS STATS MODEL HANDLING

See the BIDS stats model website for more information.

class bids.**Model**

Class to deal with BIDS stats models

See the BIDS stats model website for more information.

USAGE:: matlab

bm = bids.Model('init', true, ...
'file', path_to_bids_stats_model_file, ... 'tolerant', true, 'verbose', false);

Parameters

- **init** (logical) – if true this will initialize an empty model. Defaults to false.
- **file** (path) – fullpath the JSON file containing the BIDS stats model
- **tolerant** (logical) – turns errors into warning
- **verbose** (logical) – silences warnings

6.1 Examples

```
% initialize and write an empty model
bm = bids.Model('init', true);
filename = fullfile(pwd, 'model-foo_smdl.json');
bm.write(filename);
```

```
% load a stats model from a file
model_file = fullfile(get_test_data_dir(), ...
                      '..', ...
                      'data', ...
                      'model', ['model-narps_smdl.json']);

bm = bids.Model('file', model_file, 'verbose', false);
```

Property Summary

content

"raw" content of a loaded JSON

Name
Name of the model

Description
Description of the model

BIDSModelVersion
Version of the model

Input
Input of the model

Nodes
Nodes of the model

Edges
Edges of the model

tolerant
if `true` turns error into warning

verbose
hides warning if `false`

dag_built
if the directed acyclic graph has been built

Method Summary

get_nodes(*varargin*)
Get a specific node from the model given its Level and / or Name.

USAGE:

```
[value, idx] = bm.get_nodes('Level', '', 'Name', '')
```

Parameters
- **Level** – Must be one of `Run`, `Session`, `Subject`, `Dataset`. Default to `''`
- **Name** – Default to `''`

Returns: value - Node(s) as struct if there is only one or a cell if more
idx - Node index

6.1.1 Example

```
bm = bids.Model('file', model_file('narps'), 'verbose', false);

% Get all nodes
bm.get_nodes()

% Get run level node
bm.get_nodes('Level', 'Run')

% Get the "Negative" node
bm.get_nodes('Name', 'negative')
```

get_edge(*field*, *value*)

USAGE:

```
edge = bm.get_edges(field, value)
```

field can be any of {'Source', 'Destination'}

get_edges_from_nodes()

Generates all the default edges from the list of nodes in the model.

USAGE:

```
bm = bm.get_edges_from_nodes();
edges = bm.Edges();
```

validate()

Very light validation of fields that were not checked on loading. Automatically run on loaoding of a dataset.

USAGE:

```
bm.validate()
```

validate_edges()

USAGE:

```
bm.validate_edges()
```

get_transformations(*varargin*)

USAGE:

```
transformations = bm.get_transformations('Name', 'node_name')
```

Parameters
Name (char) – name of the node whose transformations we want

get_dummy_contrasts(*varargin*)

USAGE:

```
dummy_contrasts = bm.get_dummy_contrasts('Name', 'node_name')
```

Parameters
Name (char) – name of the node whose dummy contrasts we want

get_contrasts(*varargin*)

USAGE:

```
contrasts = bm.get_contrasts('Name', 'node_name')
```

Parameters
Name (char) – name of the node whose contrasts we want

get_model(*varargin*)

USAGE:

```
model = bm.get_model('Name', 'node_name')
```

Parameters
Name (char) – name of the node whose model we want

get_design_matrix(*varargin*)

USAGE:

```
matrix = bm.get_design_matrix('Name', 'node_name')
```

Parameters
Name (char) – name of the node whose model matrix we want

default(*varargin*)

Generates a default BIDS stats model for a given data set

USAGE:

```
bm = bm.default(BIDS, tasks)
```

Parameters
- **BIDS** (path or structure) – fullpath to a BIDS dataset or output structure from bids.layout
- **tasks** (char or cellstr) – tasks to include in the model

6.1.2 Example

```
pth_bids_example = get_test_data_dir();
BIDS = bids.layout(fullfile(pth_bids_example, 'ds003'));
bm = bids.Model();
bm = bm.default(BIDS, 'rhymejudgement');
filename = fullfile(pwd, 'model-rhymejudgement_smdl.json');
bm.write(filename);
```

update()

Update content for writing

USAGE:

```
bm = bm.update()
```

write(*filename*)

USAGE:

```
bm.write(filename)
```

static empty_node(*level*)

USAGE:

```
node = Model.empty_node('run')
```

static empty_transformations()

USAGE:

```
transformations = Model.empty_transformations()
```

static empty_model()

USAGE:

```
model = Model.empty_model()
```

CHAPTER

SEVEN

VARIABLE TRANSFORMATIONS

Those transformers are meant to be used to manipulate the content of TSV files once loaded as structure with `bids.util.tsvread`.

They are mostly meant to be used to implement the transformations described in BIDS stats models but can also be used to manipulate TSV files in batches.

More information on how they function can be found in the variable-transform repository.

The behavior and their "call" in JSON should (hopefully) be fairly close to the pybids-transformers.

7.1 Applying transformations

An "array" of transformations can be applied one after the other using `bids.transformers()`.

`bids.`**`transformers`**`(`*`varargin`*`)`

Apply transformers to a structure.

USAGE:

```
new_content = transformers(trans, data)
```

Parameters

- **`transformers`** (`structure`)
- **`data`** (`structure`)

Returns

- **new_content**
  (structure)
- **json**
  (structure) json equivalent of the transformers

7.1.1 Example

```
data = bids.util.tsvread(path_to_tsv);

% load transformation instruction from a model file
bm = bids.Model('file', model_file);
transformers = bm.get_transformations('Level', 'Run');

% apply transformers
new_content = bids.transformers(transformers.Instructions, data);

% if all fields in the structure have the same number of rows one
% create a new tsv file
bids.util.tsvwrite(path_to_new_tsv, new_content)
```

See also: bids.Model

7.2 Basic operations

- Add
- Subtract
- Multiply
- Divide
- Power

bids.transformers_list.**Basic**(*transformer*, *data*)

Performs a basic operation with a `Value` on the `Input`

Each of these transformations takes one or more columns, and performs a mathematical operation on the input column and a provided operand. The operations are performed on each column independently.

Arguments:

Parameters

- **`Name`** – **mandatory**. Any of `Add`, `Subtract`, `Multiply`, `Divide`, `Power`.
- **`Input`** (`char or array`) – **mandatory**. A array of columns to perform operation on.
- **`Value`** (`float`) – **mandatory**. The value to perform operation with (i.e. operand).
- **`Query`** (`char`) – Optional. logical expression used to select on which rows to act.
- **`Output`** (`char or array`) – Optional. List of column names to write out to.

By default, computation is done in-place on the input (meaning that input columns are overwritten). If provided, the number of values must exactly match the number of input values, and the order will be mapped 1-to-1.

7.3 Logical operations

- And
- Or
- Not

bids.transformers_list.**Logical**(*transformer*, *data*)

Each of these transformations:

- takes 2 or more columns as input
- performs the corresponding logical operation
- inclusive or
- conjunction
- logical negation
- returning a single column as output.

If non-logical input are passed, it is expected that:

- all zero or nan (for numeric data types),
- "NaN" or empty (for char) values

will evaluate to false and all other values will evaluate to true.

Arguments:

Parameters

- **Name** – **mandatory**. Any of And, Or, Not.
- **Input** (array) – **mandatory**. An array of columns to perform operation on. Only 1 for Not
- **Output** (char or array) – Optional. The name of the output column.

7.4 Munge operations

Transformations that primarily involve manipulating/munging variables into other formats or shapes.

7.4.1 Assign

bids.transformers_list.**Assign**(*transformer*, *data*)

The Assign transformation assigns one or more variables or columns (specified as the input) to one or more other columns (specified by target and/or output as described below).

Arguments:

Parameters

- **Input** (char or array) – **mandatory**. The name(s) of the columns from which attribute values are to be drawn (for assignment to the attributes of other columns). Must exactly match the length of the target argument.

- **Target** (char or array) – **mandatory**. the name(s) of the columns to which the attribute values taken from the input are to be assigned. Must exactly match the length of the input argument. Names are mapped 1-to-1 from input to target.

Note: If no output argument is specified, the columns named in target are modified in-place.

Parameters

Output (char or array) – Optional. Names of the columns to output the result of the assignment to. Must exactly match the length of the input and target arguments.

If no output array is provided, columns named in target are modified in-place.

If an output array is provided:

- each column in the target array is first cloned,
- then the reassignment from the input to the target is applied;
- finally, the new (cloned and modified) column is written out to the column named in output.

Parameters

- **InputAttr** (char or array) – Optional. Specifies which attribute of the input column to assign. Defaults to value. If a array is passed, its length must exactly match that of the input and target arrays.
- **TargetAttr** (char or array) – Optional. Specifies which attribute of the output column to assign to. Defaults to value. If a array is passed, its length must exactly match that of the input and target arrays.

InputAttr and TargetAttr must be one of:

- value,
- onset,
- or duration.

Note: This transformation is non-destructive with respect to the input column(s). In case where in-place assignment is desired (essentially, renaming a column), either use the rename transformation, or set output to the same value as the input.

To reassign the value property of a variable named response_time to the duration property of a face variable (as one might do in order to, e.g., model trial-by-trial reaction time differences for a given condition using a varying-epoch approach), and write it out as a new face_modulated_by_RT column.

7.4.2 Concatenate

bids.transformers_list.**Concatenate**(*transformer*, *data*)

Concatenate columns together.

Arguments:

Parameters

- **Input** (array) – **mandatory**. Column(s) to concatenate. Must all be of the same length.
- **Output** (char) – Optional. Name of the output column.

7.4.3 Copy

bids.transformers_list.**Copy**(*transformer*, *data*)

Clones/copies each of the input columns to a new column with identical values and a different name. Useful as a basis for subsequent transformations that need to modify their input in-place.

Arguments:

Parameters

- **Input** (char or array) – **mandatory**. Column names to copy.
- **Output** (char or array) – Optional. Names to copy the input columns to. Must be same length as input, and columns are mapped one-to-one from the input array to the output array.

7.4.4 Delete

bids.transformers_list.**Delete**(*transformer*, *data*)

Deletes column(s) from further analysis.

Arguments:

Parameters

Input (char or array) – **mandatory**. The name(s) of the columns(s) to delete.

Note: The Select transformation provides the inverse function (selection of columns to keep for subsequent analysis).

7.4.5 DropNA

bids.transformers_list.**Drop_na**(*transformer*, *data*)

Drops all rows with "n/a".

Arguments:

Parameters

- **Input** (char or array) – **mandatory**. The name of the variable to operate on.
- **Output** (char or array) – Optional. The column names to write out to. By default, computation is done in-place meaning that input columnise overwritten).

7.4.6 Factor

bids.transformers_list.**Factor**(*transformer*, *data*)

Converts a nominal/categorical variable with N unique levels to either N indicators (i.e., dummy-coding).

Arguments:

Parameters
Input (char or array) – **mandatory**. The name(s) of the variable(s) to dummy-code.

By default it is the first factor level when sorting in alphabetical order (e.g., if a condition has levels 'dog', 'apple', and 'helsinki', the default reference level will be 'apple').

The name of the output columns for 2 input columns `gender` and `age` with 2 levels (`M`, `F`) and (`20`, `30`) respectivaly will of the shape:

- `gender_F_age_20`
- `gender_F_age_20`
- `gender_M_age_30`
- `gender_M_age_30`

7.4.7 Filter

bids.transformers_list.**Filter**(*transformer*, *data*)

Subsets rows using a logical expression.

Arguments:

Parameters

- **Input** (char or array) – **mandatory**. The name(s) of the variable(s) to operate on.
- **Query** (char) – **mandatory**. logical expression used to filter

Supports:

- >, <, >=, <=, ==, ~= for numeric values
- ==, ~= for char operation (case sensitive). Regular expressions are supported

Parameters
Output (char or array) – Optional. The optional column names to write out to.

By default, computation is done in-place (i.e., input columnise overwritten). If provided, the number of values must exactly match the number of input values, and the order will be mapped 1-to-1.

7.4.8 Label identical rows

bids.transformers_list.**Label_identical_rows**(*transformer*, *data*)

Creates an extra column to index consecutive identical rows in a column. The index restarts at 1 with every change of row content. This can for example be used to label consecutive events of the same trial_type in a block.

Arguments:

Parameters

- **Input** (`char or array`) – **mandatory**. The name(s) of the variable(s) to operate on.
- **Cumulative** (`logical`) – **optional**. Defaults to `False`. If `True`, the labels are not reset to 0 when encountering new row content.

Note: The labels will be by default be put in a column called Input(i)_label

---

7.4.9 Merge identical rows

`bids.transformers_list.`**Merge_identical_rows**(*transformer*, *data*)

Merge consecutive identical rows.

Arguments:

Parameters

Input (`char or array`) – **mandatory**. The name(s) of the variable(s) to operate on.

Note:

- Only works on data commit from event.tsv
- Content is sorted by onset time before merging
- If multiple variables are specified, they are merged in the order they are specified
- If a variable is not found, it is ignored
- If a variable is found, but is empty, it is ignored
- The content of the other columns corresponds to the last row being merged: this means that the content from other columns but the one specified in will be deleted except for the last one

---

7.4.10 Replace

`bids.transformers_list.`**Replace**(*transformer*, *data*)

Replaces values in one or more input columns.

Arguments:

Parameters

- **Input** (`char or array`) – **mandatory**. Name(s of column(s) to search and replace within.
- **Replace** (`array of objects`) – **mandatory**. The mapping old values (`"key"`) to new values. (`"value"`). `key` can be a regular expression.
- **Attribute** (`array`) – Optional. The column attribute to apply the replace to.

Valid values include:

- `"value"` (the default),
- `"duration"`,
- `"onset"`,
- and `"all"`.

In the last case, all three attributes ("value", "duration", and "onset") will be scanned.

Parameters

Output (char or array) – Optional. Optional names of columns to output. Must match length of input column(s) if provided, and columns will be mapped 1-to-1 in order. If no output values are provided, the replacement transformation is applied in-place to all the inputs.

7.4.11 Select

bids.transformers_list.**Select**(*transformer*, *data*)

The select transformation specifies which columns to retain for subsequent analysis.

Any columns that are not specified here will be dropped.

The only exception is when dealing with data with onset and duration columns (from *_events.tsv files) in this case the onset and duration column are also automatically selected.

Arguments:

Parameters

Input (char or array) – **mandatory**. The names of all columns to keep. Any columns not in this array will be deleted and will not be available to any subsequent transformations or downstream analyses.

Note: one can think of select as the inverse the Delete transformation that removes all named columns from further analysis.

7.4.12 Split

bids.transformers_list.**Split**(*transformer*, *data*)

Split a variable into N variables as defined by the levels of one or more other variables.

Arguments:

Parameters

- **Input** (array) – **mandatory**. The name of the variable(s) to operate on.
- **By** (array) – Optional. Name(s) for variable(s) to split on.

For example, for given a variable Condition that we wish to split on two categorical columns A and B, where a given row has values A=a and B=1, the generated name will be Condition_BY_A_a_BY_B_1.

7.5 Compute operations

Transformations that primarily involve numerical computation on variables.

7.5.1 Constant

bids.transformers_list.**Constant**(*transformer*, *data*)

Adds a new column with a constant value (numeric or char).

Arguments:

Parameters

- **Output** (char or array) – **mandatory**. Name of the newly generated column.
- **Value** (float or char) – Optional. The value of the constant, defaults to 1.

7.5.2 Mean

bids.transformers_list.**Mean**(*transformer*, *data*)

Compute mean of a column.

JSON EXAMPLE

```
{
  "Name":  "Mean",
  "Input": "reaction_time",
  "OmitNan": false,
  "Output": "mean_RT"
}
```

Arguments:

param Input
: **mandatory**. The name of the variable to operate on.

type Input
: char or array

param OmitNan
: Optional. If false any column with nan values will return a nan value. If true nan values are skipped. Defaults to false.

type OmitNan
: logical

param Output
: Optional. The optional column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

type Output
: char or array

CODE EXAMPLE

```
transformer = struct('Name', 'Mean', ...
                     'Input', 'reaction_time', ...
                     'OmitNan', false, ...
                     'Ouput', 'mean_RT');

data.reaction_time = TODO

data = bids.transformers(transformer, data);

data.mean_RT = TODO

ans = TODO
```

7.5.3 Product

bids.transformers_list.**Product**(*transformer*, *data*)

Computes the row-wise product of two or more columns.

Arguments:

Parameters

- **Input** (array) – **mandatory**. Names of two or more columns to compute the product of.
- **Output** (string or array) – **mandatory**. Name of the newly generated column.
- **OmitNan** (logical) – Optional. If false any column with nan values will return a nan value. If true nan values are skipped. Defaults to false.

7.5.4 Scale

bids.transformers_list.**Scale**(*transformer*, *data*)

Scales the values of one or more columns.

Semantics mimic scikit-learn, such that demeaning and rescaling are treated as independent arguments, with the default being to apply both (i.e., standardizing each value so that it has zero mean and unit SD).

Arguments:

Parameters

- **Input** (char or array) – **mandatory**. Names of columns to standardize.
- **Demean** (logical) – Optional. If true, subtracts the mean from each input column (i.e., applies mean-centering).
- **Rescale** (logical) – Optional. If true, divides each column by its standard deviation.
- **ReplaceNa** (logical) – Optional. Whether/when to replace missing values with 0. If "off", no replacement is performed. If "before", missing values are replaced with 0 before scaling. If "after", missing values are replaced with 0 after scaling. Defaults to "off"

- **Output** (`char or array`) – Optional. Optional names of columns to output. Must match length of input column if provided, and columns will be mapped 1-to-1 in order. If no output values are provided, the scaling transformation is applied in-place to all the input.

7.5.5 Std

`bids.transformers_list.`**Std**(*transformer*, *data*)

Compute the sample standard deviation.

Arguments:

Parameters

- **Input** (`char or array`) – **mandatory**. The name of the variable to operate on.
- **OmitNan** (`logical`) – Optional. If `false` any column with nan values will return a nan value. If `true` nan values are skipped. Defaults to `false`.
- **Output** (`char or array`) – Optional. The optional column names to write out to. By default, computation is done in-place (i.e., input columnise overwritten).

7.5.6 Sum

`bids.transformers_list.`**Sum**(*transformer*, *data*)

Computes the (optionally weighted) row-wise sums of two or more columns.

Arguments:

Parameters

- **Input** (`array`) – **mandatory**. Names of two or more columns to sum.
- **Output** (`char or array`) – **mandatory**. Name of the newly generated column.
- **OmitNan** (`logical`) – Optional. If `false` any column with nan values will return a nan value. If `true` nan values are skipped. Defaults to `false`.
- **Weights** (`array`) – Optional. Optional array of floats giving the weights of the columns. If provided, length of weights must equal to the number of values in input, and weights will be mapped 1-to-1 onto named columns. If no weights are provided, defaults to unit weights (i.e., simple sum).

7.5.7 Threshold

`bids.transformers_list.`**Threshold**(*transformer*, *data*)

Thresholds input values at a specified cut-off and optionally binarizes the result.

Arguments:

Parameters

- **Input** (`char or array`) – **mandatory**. The name(s)of the column(s) to threshold/binarize.
- **Threshold** (`float`) – Optional. The cut-off to use for thresholding. Defaults to `0`.
- **Binarize** (`logical`) – Optional. If `true`, thresholded values will be binarized (i.e., all non-zero values will be set to 1). Defaults to `false`.

- **Above** (`logical`) – Optional. Specifies which values to retain with respect to the cut-off. If `true`, all value above the threshold will be kept; if `false`, all values below the threshold will be kept. Defaults to `true`.
- **Signed** (`logical`) – Optional. Specifies whether to treat the threshold as signed (default) or unsigned.

For example, when passing above=true and threshold=3, if signed=true, all and only values above +3 would be retained. If signed=false, all absolute values > 3 would be retained (i.e.,values in the range -3 < X < 3 would be set to 0).

Parameters

Output (`char or array`) – Optional. Optional names of columns to output. Must match length of input column if provided, and columns will be mapped 1-to-1 in order. If no output values are provided, the threshold transformation is applied in-place to all the inputs.

CHAPTER

EIGHT

USING THE BIDS SCHEMA

class bids.**Schema**

Class to interact with the BIDS schema

USAGE:

```
schema = bids.Schema(use_schema)
```

use_schema: logical

Constructor Summary

Schema(*use_schema*)

USAGE:

```
schema = bids.Schema(use_schema)
```

use_schema: logical

Method Summary

load(*use_schema*)

Load schema.

USAGE:

```
schema = bids.Schema()
schema = schema.load()
```

return_entities(*varargin*)

Return all the entities for or one or more datatype and one or more suffixes.

USAGE:

```
entities = schema.return_entities('datatypes', datatypes, ...
                                  'suffixes', suffixes, ...
                                  'required_only', false)
```

Parameters

- **datatypes** (char or cellstr) – For example 'func'.
- **suffixes** (char or cellstr) – For example 'bold'.
- **required_only** (logical) – If true, only the required entities are returned.

8.1 Example

```
suffixes = schema.return_entities('datatypes', 'func')
suffixes = schema.return_entities('datatypes', 'func', 'suffixes', 'bold')
suffixes = schema.return_suffixes('datatypes', {'anat', 'func'}, ...
                                  'suffixes', {'T1w', 'bold'}, ...
                                  'required_only', true)
```

return_modalities(*subject*, *modality_group*)

Return the datatypes for a given for a given modality group for a given subject. For example, "mri" will give: "func", "anat", "dwi", "fmap"…

USAGE:

```
modalities = schema.return_modalities(subject, modality_group)
```

Parameters

- **subject** (struct) – Subject information: subject.path, … See parse_subject subfunction for layout.m for details.
- **modality_group** (char) – Any of the BIDS modality

If we go schema-less, we list directories in the subject/session folder as proxy of the datatypes that we have to parse.

entity_order(*varargin*)

Return the 'correct' order for entities of entity list.

If there are non BIDS entities they are added after the BIDS ones in alphabetical order.

USAGE:

```
order = schema.entity_order(entity_list, 'use_short_form', true)
```

8.2 Example

```
schema = bids.Schema();

%  get the order of all the BIDS entities
order = schema.entity_order()

% reorder typical BIDS entities
entity_list_to_order = {'description'
                        'run'
                        'subject'};
order = schema.entity_order(entity_list_to_order)

    {'subject'
     'run'
     'description'};

% reorder non-BIDS and typical BIDS entities
entity_list_to_order = {'description'
```

(continues on next page)

(continued from previous page)

```
                        'run'
                        'foo'
                        'subject'};
order = schema.entity_order(entity_list_to_order)

    {'subject'
     'run'
     'description'
     'foo'};
```

return_entity_name(*entity_key*)

Return the name of an entity key.

USAGE:

```
entity_name = schema.return_entity_key(entity_key)
```

8.3 Example

```
key = schema.return_entity_key('desc')

    'description'


key = schema.return_entity_key({'desc', 'sub'})

    {'description'; 'subject'}
```

return_entity_key(*entity_names*)

Return the key of an entity.

USAGE:

```
key = schema.return_entity_key(entity)
```

8.4 Example

```
key = schema.return_entity_key('description')

    'desc'
```

return_modality_groups()

USAGE:

```
groups = schema.return_modality_groups()
```

Returns a dummy variable if we go schema less

list_suffix_groups(*datatype*, *scope*)

Creates a structure of all the suffix group of a datatype.

USAGE:

```
suffix_groups = schema.list_suffix_groups(datatype, scope)
```

Parameters
- **datatype** (char) – for example 'func'
- **scope** (char) – 'raw' or 'derivatives' or 'all'

return_suffix_groups_for_datatype(*datatype*)

Returns a structure of all the suffix group of a datatype.

USAGE:

```
suffix_groups = schema.return_suffix_groups_for_datatype(datatype)
```

Parameters
datatype (char)

8.5 Example

```
suffix_groups = schema.return_suffix_groups_for_datatype('func')
```

return_entities_for_suffix_group(*suffix_group*)

Entities are returned in the expected order according to the schema.

USAGE:

```
entities = schema.return_entities_for_suffix_group(suffix_group)
```

Parameters
suffix_group (struct)

8.6 Example

```
suffix_groups = return_suffix_groups_for_datatype(obj, datatype)
entities = schema.return_entities_for_suffix_group(suffix_groups(1))
```

required_entities_for_suffix_group(*this_suffix_group*)

Return a logical vector to track which entities of a suffix group are required in the bids schema.

USAGE:

```
required_entities = schema.required_entities_for_suffix_group(this_suffix_
↪group)
```

Parameters
this_suffix_group (struct)

find_suffix_group(*modality*, *suffix*)

For a given suffix and modality, this return the "suffix group" this suffix belongs to.

USAGE:

```
suffix_group = schema.find_suffix_group(modality, suffix)
```

Parameters
- **modality** (char)
- **suffix** (char)

8.7 Example

```
schema = bids.Schema();
suffix_group = schema.find_suffix_group('anat', 'T1w');
suffix_group

    'nonparametric'
```

return_suffixes(*varargin*)

Return all the suffixes for or one or more datatype.

USAGE:

```
suffixes = schema.return_suffixes('datatypes', datatypes)
```

8.8 Example

```
suffixes = schema.return_suffixes('datatypes', 'func')

suffixes = schema.return_suffixes('datatypes', {'anat', 'func'})
```

return_datatypes_for_suffix(*suffix*)

For a given suffix, returns all the possible datatypes that have this suffix.

Parameters
suffix (char)

8.9 Example

```
schema = bids.Schema();
datatypes = schema.return_datatypes_for_suffix('bold');
assertEqual(datatypes, {'func'});
```

return_entities_for_suffix_modality(*suffix*, *modality*)

returns the list of entities for a given suffix of a given modality

USAGE:

```
[entities, required] = schema.return_entities_for_suffix_modality(suffix,␣
↪modality)
```

Parameters
- **modality** (char)
- **suffix** (char)

return_modality_suffixes_regex(*modality*)

creates a regular expression of suffixes for a given imaging modality

USAGE:

```
reg_ex = schema.return_modality_suffixes_regex(modality)
```

Parameters
modality (char)

return_modality_extensions_regex(*modality*)

creates a regular expression of extensions for a given imaging modality

USAGE:

```
reg_ex = schema.return_modality_regex(modality)
```

Parameters
modality (char)

return_modality_regex(*modality*)

creates a regular expression of suffixes and extension for a given imaging modality

Parameters
modality (char)

get_definition(*word*)

finds definition of a column header in a the BIDS schema

USAGE:

```
[def, status] = schema.get_definition(word)
```

static ci_check(*variable_to_check*)

Mostly to avoid some crash in continuous integration

CHAPTER

NINE

PERFORMANCE

bids-matlab's performance may vary depending on which options you choose for indexing with *bids.layout()*.

Relying on the BIDS schema (`'use_schema', true`) to only include files that comply with it may slow down performance a bit.

Index dependencies (`'index_dependencies', true`) to detect files with explicit dependencies between them (for example when a fieldmap mentions that is intended for a specific bold run) is much slower and you may consider setting this to `false` when if you know that you have no such dependencies in your dataset.

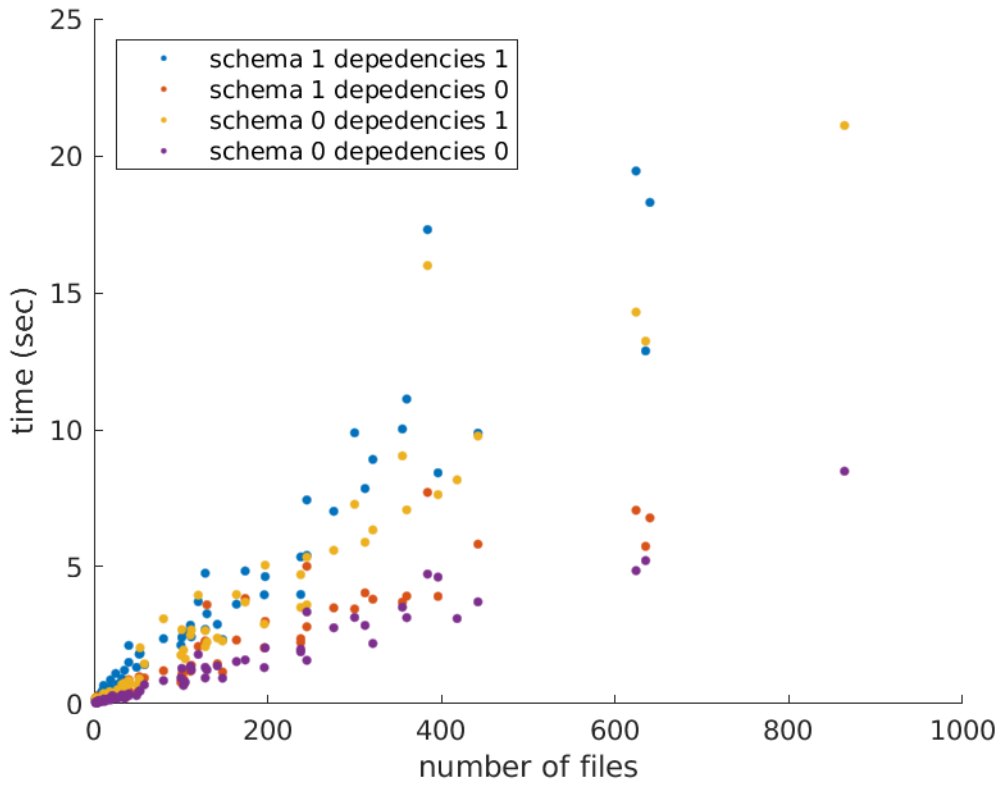

Fig. 1: layout indexing performance depending on chosen options

CHAPTER

TEN

DEVELOPER DOCUMENTATION

10.1 +internal

bids.internal.**add_missing_field**(*structure*, *field*)

USAGE:

```
structure = add_missing_field(structure, field)
```

bids.internal.**append_to_layout**(*file*, *subject*, *modality*, *schema*, *previous*)

appends a file to the BIDS layout by parsing it according to the provided schema

USAGE:

```
subject = append_to_layout(file, subject, modality, schema == [])
```

Parameters

- **file** (char)
- **subject** (structure) – subject sub-structure from the BIDS layout
- **modality** (char)
- **schema** (structure)

bids.internal.**camel_case**(*str*)

Removes non alphanumeric characters and uppercase first letter of all words but the first

USAGE:

```
str = camel_case(str)
```

Parameters

str (char)

Returns

str

(char) returns the input with an upper case for first letter for all words but the first one (camelCase) and removes invalid characters (like spaces).

bids.internal.**create_unordered_list**(*list*)

turns a cell string or a structure into a string that is an unordered list to print to the screen

USAGE:

```
list = bids.internal.create_unordered_list(list)
```

Parameters

list (cell string or structure) – obligatory argument.

bids.internal.**download**(*URL*, *output_dir*, *verbose*)

USAGE:

```
filename = download(URL, output_dir, verbose)
```

bids.internal.**ends_with**(*str*, *pattern*)

Checks id character array 'str' ends with 'pat'

USAGE:

```
res = bids.internal.endsWith(str, pattern)
```

Parameters

- **str** (character array)
- **pattern** (character array)

Based on the equivalent function from SPM12

bids.internal.**error_handling**(*varargin*)

USAGE:

```
error_handling(function_name, id, msg, tolerant, verbose)
```

Parameters

- **function_name** – default = bidsMatlab
- **id** (char) – default = unspecified
- **msg** (char) – default = unspecified
- **tolerant** (logical)
- **verbose** (logical)

10.1.1 Example

```
bids.internal.error_handling(mfilename(), 'thisError', 'this is an error', tolerant,
↪ verbose)
```

bids.internal.**file_utils**(*str*, *varargin*)

Character array (or cell array of char) handling facility

USAGE:

To list files or directories (with fullpath if necessary):

```
[files, dirs] = bids.internal.file_utils('List',   directory,        regexp)
[files, dirs] = bids.internal.file_utils('FPList', directory,        regexp)
[dirs]        = bids.internal.file_utils('List',   directory, 'dir', regexp)
[dirs]        = bids.internal.file_utils('FPList', directory, 'dir', regexp)
```

To get a certain piece of information from a file:

```
str = bids.internal.file_utils(str, option)
```

str - character array, or cell array of char

option - char of requested item - one among:
{'path', 'basename', 'ext', 'filename', 'cpath', 'fpath'}

To set a certain piece of information from a file:

```
str = bids.internal.file_utils(str, opt_key, opt_val, ...)
```

str - character array, or cell array of char

opt_key - char of targeted item - one among:
{'path', 'basename', 'ext', 'filename', 'prefix', 'suffix'}

opt_val - char of new value for feature

Based on spm_file.m and spm_select.m from SPM12.

bids.internal.**format_path**(*pth*)

USAGE:

```
pth = bids.internal.format_path(pth)
```

Replaces single '' by '/' in Windows paths to prevent escaping warning when printing a path to screen

Parameters
pth (char or cellstr$) – If pth is a cellstr of paths, pathToPrint will work recursively on it.

bids.internal.**get_meta_list**(*varargin*)

Read a BIDS's file metadata according to the inheritance principle

USAGE:

```
metalist = bids.internal.get_metadata(filename, ...
                                      pattern, ...
                                      use_inheritance)
```

Parameters

- **filename** (char) – fullpath name of file following BIDS standard
- **pattern** (char) – Regular expression matching the metadata file default = '^.*%s\\.json$' If provided, it must at least be '%s'.

Returns
metalist - list of paths to metafiles

bids.internal.**get_metadata**(*metafile*)

Read a BIDS's file metadata according to the inheritance principle.

USAGE:

```
meta = bids.internal.get_metadata(metafile)
```

Parameters
metafile (char or array of chars) – list of fullpath names of metadata files.

Returns

- **meta**
  metadata structure

bids.internal.**get_version**()

Reads the version number of the pipeline from the txt file in the root of the repository.

USAGE:

```
version_number = bids.internal.get_version()
```

Returns

version_number
(char) Use semantic versioning format (like v0.1.0)

bids.internal.**is_github_ci**()

(C) Copyright 2021 Remi Gau

bids.internal.**is_octave**()

Returns true if the environment is Octave.

USAGE:

```
status = bids.internal.is_octave()
```

Returns

status
(logical)

bids.internal.**is_valid_fieldname**(*some_str*)

A valid MATLAB identifier is a character vector of (A-Z, a-z, 0-9) and underscores, such that the first character is a letter and the length of the character vector is less than or equal to namelengthmax.

USAGE:

```
status = is_valid_fieldname(some_str)
```

bids.internal.**keep_file_for_query**(*file_struct*, *options*)

USAGE:

```
status = keep_file_for_query(file_struct, options)

returns ``false`` if the file is to be kept when running ``bids.query``
```

bids.internal.**list_all_trial_types**(*varargin*)

List all the trial_types in all the events.tsv files for a task.

USAGE:

```
trial_type_list = bids.internal.list_all_trial_types(BIDS, , ...
                                                     task, ...
                                                     'trial_type_col', 'trial_type',
↪ ...
                                                     'tolerant', true, ...
                                                     'verbose', false)
```

Parameters

- **BIDS** (structure or char) – BIDS directory name or BIDS structure (from bids.layout)
- **task** (char) – name of the task
- **trial_type_col** (char) – Optional. Name of the column containing the trial type. Defaults to 'trial_type'.
- **tolerant** (logical) – Optional. Default to true.
- **verbose** (logical) – Optional. Default to false.

bids.internal.**match_structure_fields**(*struct_one*, *struct_two*)

Update list of fields of a structure so it matches that of another.

USAGE:

```
[struct_one, struct_two] = match_structure_fields(struct_one, struct_two)
```

bids.internal.**parse_filename**(*filename*, *fields*, *tolerant*, *verbose*)

Split a filename into its building constituents

USAGE:

```
p = bids.internal.parse_filename(filename, fields, tolerant, verbose)
```

Parameters

- **filename** (string) – filename to parse that follows the pattern sub-label[_entity-label]*_suffix.extension
- **fields** (cell) – cell of strings of the entities to use for parsing

10.1.2 Example

```
filename = '../sub-16/anat/sub-16_ses-mri_run-1_acq-hd_T1w.nii.gz';

bids.internal.parse_filename(filename)

ans =

struct with fields:

  'filename', 'sub-16_ses-mri_run-1_acq-hd_T1w.nii.gz', ...
  'suffix', 'T1w', ...
  'ext', '.nii.gz', ...
  'entities', struct('sub', '16', ...
                     'ses', 'mri', ...
                     'run', '1', ...
                     'acq', 'hd');
```

10.1.3 Example

```
filename = '../sub-16/anat/sub-16_ses-mri_run-1_acq-hd_T1w.nii.gz';
fields = {'sub', 'ses', 'run', 'acq', 'ce'};
output = bids.internal.parse_filename(filename, fields);
```

The output will have the following shape:

```
output = struct( ...
                 'filename', 'sub-16_ses-mri_run-1_acq-hd_T1w.nii.gz', ...
                 'suffix', 'T1w', ...
                 'ext', '.nii.gz', ...
                 'entities', struct('sub', '16', ...
                                    'ses', 'mri', ...
                                    'run', '1', ...
                                    'acq', 'hd', ...
                                    'ce', ''), ...
                 'prefix', '');
```

bids.internal.**plot_diagnostic_table**(*diagnostic_table*, *headers*, *yticklabel*, *fig_name*, *visible*)

Plot a diagnostic table to see the number of files per subject or of trials per run.

USAGE:

```
plot_diagnostic_table(diagnostic_table, headers, yticklabel, fig_name)
```

Parameters

- **diagnostic_table** (`n X m array of integers`) – table to plot
- **headers** (`n X 1 cell of struct`) – Used to created the column names
- **yticklabel** (`m X 1 cellstr`)
- **fig_name** (`str`)

bids.internal.**regexify**(*string*)

Turns a string into a simple regex. Useful to query bids dataset with bids.query that by default expects will treat its inputs as regexp.

Input –> Output

`foo` –> `^foo$`

USAGE:

```
string = bids.internal.regexify(string)
```

bids.internal.**replace_placeholders**(*boilerplate_text*, *metadata*)

bids.internal.**return_file_index**(*BIDS*, *modality*, *filename*)

For a given filename and modality, it returns the file index in the subject sub-structure of the BIDS structure.

UISAGE:

```
file_idx = return_file_index(BIDS, modality, filename)
```

bids.internal.**return_file_info**(*BIDS*, *fullpath_filename*)

USAGE:

```
file_info = return_file_info(BIDS, fullpath_filename)
```

bids.internal.**return_subject_index**(*BIDS*, *filename*)

For a given filename, it returns the subject index in BIDS structure so that: `BIDS.subjects(sub_idx)`

USAGE:

```
sub_idx = return_subject_index(BIDS, filename)
```

bids.internal.**root_dir**()

bids.internal.**starts_with**(*str*, *pattern*)

Checks id character array 'str' starts with 'pattern'

USAGE:

```
res = bids.internal.startsWith(str, pattern)
```

Parameters

- **str** (character array)
- **pattern** (character array)

Based on the equivalent function from SPM12.

bids.internal.**url**(*section*)

returns URL of some specific sections of the spec

USAGE:

```
value = url(section)
```

CHAPTER

ELEVEN

CHANGELOG

All notable changes to this project will be documented in this file.

The format is based on Keep a Changelog, and this project adheres to Semantic Versioning.

11.1 [Unreleased]

11.1.1 Added

- [ENH] Add zero padding when numbers are passed for indices to `bids.File` or `bids.File.rename` #680 by @Remi-Gau

11.1.2 Changed

11.1.3 Deprecated

11.1.4 Removed

11.1.5 Fixed

- [FIX] Create valid participants and sessions tsv during dataset init #688 by @Remi-Gau

11.1.6 Security

11.2 [v0.2.0]

11.2.1 Changed

11.2.2 Deprecated

11.2.3 Removed

11.2.4 Fixed

11.2.5 Security

11.3 [v0.1.0]

11.3.1 Changed

11.3.2 Deprecated

11.3.3 Removed

11.3.4 Fixed

11.3.5 Security

CHAPTER

TWELVE

INDICES AND TABLES

- genindex

MATLAB MODULE INDEX

b

bids, 37
bids.transformers_list, 38
bids.util, 23

INDEX

A

add_missing_field() (*in module bids.internal*), 57
append() (*bids.Description method*), 18
append_to_layout() (*in module bids.internal*), 57
Assign() (*in module bids.transformers_list*), 39

B

Basic() (*in module bids.transformers_list*), 38
bids (*module*), 5, 9, 17, 31, 37, 49
bids.transformers_list (*module*), 38
bids.util (*module*), 23
bids_path (*bids.File attribute*), 11
BIDSModelVersion (*bids.Model attribute*), 32

C

camel_case() (*in module bids.internal*), 57
check_required_entities() (*bids.File method*), 14
ci_check() (*bids.Schema static method*), 54
Concatenate() (*in module bids.transformers_list*), 41
Constant() (*in module bids.transformers_list*), 45
content (*bids.Description attribute*), 18
content (*bids.Model attribute*), 31
Copy() (*in module bids.transformers_list*), 41
copy_to_derivative() (*in module bids*), 18
create_data_dict() (*in module bids.util*), 23
create_participants_tsv() (*in module bids.util*), 23
create_readme() (*in module bids.util*), 24
create_scans_tsv() (*in module bids.util*), 24
create_sessions_tsv() (*in module bids.util*), 24
create_unordered_list() (*in module bids.internal*), 57

D

dag_built (*bids.Model attribute*), 32
default() (*bids.Model method*), 34
Delete() (*in module bids.transformers_list*), 41
derivatives_json() (*in module bids*), 17
Description (*bids.Model attribute*), 32
Description (*class in bids*), 17
Description() (*bids.Description method*), 18
diagnostic() (*in module bids*), 20
download() (*in module bids.internal*), 58
download_ds() (*in module bids.util*), 25
Drop_na() (*in module bids.transformers_list*), 41

E

Edges (*bids.Model attribute*), 32
empty_model() (*bids.Model static method*), 35
empty_node() (*bids.Model static method*), 34
empty_transformations() (*bids.Model static method*), 34
ends_with() (*in module bids.internal*), 58
entities (*bids.File attribute*), 11
entity_order (*bids.File attribute*), 11
entity_order() (*bids.Schema method*), 50
entity_required (*bids.File attribute*), 11
error_handling() (*in module bids.internal*), 58
extension (*bids.File attribute*), 11

F

Factor() (*in module bids.transformers_list*), 42
File (*class in bids*), 9
file_utils() (*in module bids.internal*), 58
filename (*bids.File attribute*), 11
Filter() (*in module bids.transformers_list*), 42
find_suffix_group() (*bids.Schema method*), 52
format_path() (*in module bids.internal*), 59

G

get_contrasts() (*bids.Model method*), 33
get_definition() (*bids.Schema method*), 54
get_design_matrix() (*bids.Model method*), 34
get_dummy_contrasts() (*bids.Model method*), 33
get_edge() (*bids.Model method*), 32
get_edges_from_nodes() (*bids.Model method*), 33
get_entity_order_from_schema() (*bids.File method*), 14
get_meta_list() (*in module bids.internal*), 59
get_metadata() (*in module bids.internal*), 60
get_modality() (*bids.File method*), 15
get_modality_from_schema() (*bids.File method*), 14
get_model() (*bids.Model method*), 33
get_nodes() (*bids.Model method*), 32
get_required_entities() (*bids.File method*), 14

get_transformations() (*bids.Model method*), 33
get_version() (*in module bids.internal*), 60

I

init() (*in module bids*), 17
Input (*bids.Model attribute*), 32
is_derivative (*bids.Description attribute*), 18
is_github_ci() (*in module bids.internal*), 60
is_octave() (*in module bids.internal*), 60
is_valid_fieldname() (*in module bids.internal*), 60

J

json_filename (*bids.File attribute*), 11
jsondecode() (*in module bids.util*), 26
jsonencode() (*in module bids.util*), 26

K

keep_file_for_query() (*in module bids.internal*), 60

L

Label_identical_rows() (*in module bids.transformers_list*), 42
layout() (*in module bids*), 5
list_all_trial_types() (*in module bids.internal*), 61
list_suffix_groups() (*bids.Schema method*), 51
load() (*bids.Schema method*), 49
Logical() (*in module bids.transformers_list*), 39

M

match_structure_fields() (*in module bids.internal*), 61
Mean() (*in module bids.transformers_list*), 45
Merge_identical_rows() (*in module bids.transformers_list*), 43
metadata (*bids.File attribute*), 11
metadata_add() (*bids.File method*), 14
metadata_append() (*bids.File method*), 14
metadata_files (*bids.File attribute*), 11
metadata_remove() (*bids.File method*), 14
metadata_update() (*bids.File method*), 14
metadata_write() (*bids.File method*), 14
mkdir() (*in module bids.util*), 26
modality (*bids.File attribute*), 11
Model (*class in bids*), 31

N

Name (*bids.Model attribute*), 31
Nodes (*bids.Model attribute*), 32
normalize_entities() (*bids.File method*), 13

P

parse_filename() (*in module bids.internal*), 61
path (*bids.File attribute*), 11
pipeline (*bids.Description attribute*), 18
plot_diagnostic_table() (*in module bids.internal*), 62
plot_events() (*in module bids.util*), 27
prefix (*bids.File attribute*), 11
Product() (*in module bids.transformers_list*), 46

Q

query() (*in module bids*), 6

R

regexify() (*in module bids.internal*), 62
rename() (*bids.File method*), 12
reorder_entities() (*bids.File method*), 12
Replace() (*in module bids.transformers_list*), 43
replace_placeholders() (*in module bids.internal*), 63
report() (*in module bids*), 19
required_entities_for_suffix_group() (*bids.Schema method*), 52
return_datatypes_for_suffix() (*bids.Schema method*), 53
return_entities() (*bids.Schema method*), 49
return_entities_for_suffix_group() (*bids.Schema method*), 52
return_entities_for_suffix_modality() (*bids.Schema method*), 53
return_entity_key() (*bids.Schema method*), 51
return_entity_name() (*bids.Schema method*), 51
return_file_index() (*in module bids.internal*), 63
return_file_info() (*in module bids.internal*), 63
return_modalities() (*bids.Schema method*), 50
return_modality_extensions_regex() (*bids.Schema method*), 54
return_modality_groups() (*bids.Schema method*), 51
return_modality_regex() (*bids.Schema method*), 54
return_modality_suffixes_regex() (*bids.Schema method*), 54
return_subject_index() (*in module bids.internal*), 63
return_suffix_groups_for_datatype() (*bids.Schema method*), 52
return_suffixes() (*bids.Schema method*), 53
root_dir() (*in module bids.internal*), 63

S

Scale() (*in module bids.transformers_list*), 46
schema (*bids.File attribute*), 11
Schema (*class in bids*), 49
Schema() (*bids.Schema method*), 49
Select() (*in module bids.transformers_list*), 44
set_derivative() (*bids.Description method*), 18
set_field() (*bids.Description method*), 18
Split() (*in module bids.transformers_list*), 44
starts_with() (*in module bids.internal*), 63

Std() (*in module bids.transformers_list*), 47
suffix (*bids.File attribute*), 11
Sum() (*in module bids.transformers_list*), 47

T

Threshold() (*in module bids.transformers_list*), 47
tolerant (*bids.Model attribute*), 32
transformers() (*in module bids*), 37
tsvread() (*in module bids.util*), 27
tsvwrite() (*in module bids.util*), 28

U

update() (*bids.File method*), 12
update() (*bids.Model method*), 34
url() (*in module bids.internal*), 63
use_schema() (*bids.File method*), 13

V

validate() (*bids.Model method*), 33
validate() (*in module bids*), 20
validate_edges() (*bids.Model method*), 33
validate_entities() (*bids.File method*), 14
verbose (*bids.Model attribute*), 32

W

write() (*bids.Description method*), 18
write() (*bids.Model method*), 34