bids-matlab

Release v0.1.0dev

BIDS-MATLAB developers

Mar 05, 2022

CONTENT

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

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 formated according to the BIDS standard

USAGE:

```
BIDS = bids.layout(pwd, ...
                   'use_schema', true, ...
                   'index_derivatives', false, ...
                   'tolerant', true, ...
                   'verbose', false)
```

Parameters

- **root** (`string`) – directory of the dataset formated according to BIDS [default: `pwd`]
- **use_schema** (`boolean`) – 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.
- **index_derivatives** (`boolean`) – if `true` this will index the content of the any `derivatives` folder in the BIDS dataset.
- **tolerant** (`boolean`) – Set to `true` to turn validation errors into warnings
- **verbose** (`boolean`) – Set to `true` to get more feedback

(C) Copyright 2016-2018 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

(C) Copyright 2018 BIDS-MATLAB developers

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

Queries a directory structure formatted according to the BIDS standard

USAGE:

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

Parameters

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

Type of query allowed:

- 'sessions'
- 'subjects'
- 'modalities'
- 'tasks'
- 'runs'
- 'spaces'
- 'labels'
- 'descriptions'
- 'resolutions'
- 'suffixes'
- 'entities'
- 'data'
- 'metadata'
- 'metafiles'
- 'dependencies'
- 'extensions'
- 'prefixes'

Warning: Note that all the query types are plurals.

Queries can "filtered" by passing more arguments key-value pairs as a list of strings or as a cell or a structure.

Note that for the entities listed below can be queried using integers:

- 'run'
- 'flip'
- 'inv'
- 'split'
- 'echo'

It is also possible to use regular expressions in the value.

Regex example:

```
% The following 2 will return the same thing
data = bids.query(BIDS, 'data', 'sub', '01')
data = bids.query(BIDS, 'data', 'sub', '^01$')

% But the following would return all the data for all subjects
% whose label ends in '01'
data = bids.query(BIDS, 'data', 'sub', '.*01')
```

—

Example 1:

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

Example 2:

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

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

Example 3:

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

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

(C) Copyright 2016-2018 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

(C) Copyright 2018 BIDS-MATLAB developers

CHAPTER

TWO

BIDS FILE HANDLING

class +bids.**File**(*varargin*)

Class to deal with BIDS files and to help to create BIDS valid names

USAGE:

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

Parameters

- **input** (filename or structure) –
- **use_schema** (boolean) –
- **tolerant** (boolean) – turns errors into warning
- **verbose** (boolean) – silences warnings

Initiliaze with a filename

EXAMPLE:

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

Initialize with a structure

EXAMPLE:

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

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

EXAMPLE:

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

(continues on next page)

(continued from previous page)

```
file.entities.desc = 'preproc';
disp(file.filename)
```

Use the BIDS schema to get entities in the right order

EXAMPLE:

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

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

(C) Copyright 2021 BIDS-MATLAB developers

prefix = "''"
bids prefix

extension = "''"
file extension

suffix = "''"
file suffix

entities = 'struct([])'
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 = '{}'
Required entities

entity_order = '{}'
Expected order of entities

schema = '[]'
BIDS schema used

update()
excuted automatically before getting a value

reorder_entities(*entity_order*)

USAGE:

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

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

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();
```

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()
```

CHAPTER

THREE

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

(C) Copyright 2018 BIDS-MATLAB developers

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

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

USAGE:

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

Parameters

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

(C) Copyright 2021 BIDS-MATLAB developers

class +bids.**Description**(*pipeline*, *BIDS*)

Class to deal with dataset_description files.

USAGE:

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

Parameters

- **pipeline** (string) – pipeline name
- **BIDS** (structure) – output from BIDS layout to identify the source dataset used when creating a derivatives dataset

(C) Copyright 2021 BIDS-MATLAB developers

content = None
dataset description content

is_derivative = 'false'
boolean

pipeline = "''"
name of the pipeline used to generate this derivative dataset

Description(*pipeline*, *BIDS*)
Instance constructor

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, returning layout of new derivatives folder

USAGE:

```
bids.copy_to_derivative(BIDS, ...
                        'pipeline_name', '', ...
                        'out_path', '', ...
                        'filters', struct(), ...
```

(continues on next page)

(continued from previous page)

```
                        'unzip', true, ...
                        'force', false, ...
                        'skip_dep', false, ...
                        'use_schema, use_schema, ...
                        'verbose', true);
```

Parameters

- **BIDS** (structure or string) – BIDS directory name or BIDS structure (from bids.layout)
- **pipeline_name** (string) – name of pipeline to use
- **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)
- **unzip** (boolean) – If true then all .gz files will be unzipped after being copied.
- **force** (boolean) – If set to false it will not overwrite any file already present in the destination.
- **skip_dep** (boolean) – If set to false it will copy all the dependencies of each file.
- **use_schema** (boolean) – If set to true it will only copy files that are BIDS valid.
- **verbose** (boolean) –

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.

(C) Copyright 2021 BIDS-MATLAB developers

+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** (string 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** (string) – Folder where the report should be printed. If empty (default) then the output is sent to the prompt.
- **read_nifti** (boolean) – 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** (boolean) – If set to false (default) the function does not output anything to the prompt.

(C) Copyright 2018 BIDS-MATLAB developers

+bids.**validate**(*root*)

BIDS Validator

USAGE:

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

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/

(C) Copyright 2018 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

(C) Copyright 2018 BIDS-MATLAB developers

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

Creates a summary figure listing the number of files for each subject / session and and imaging modality (possiblty split by task)

USAGE:

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

Parameters

- **BIDS** (structure or string) – 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** (boolean) – 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)

Examples:

```
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);
```

(C) Copyright 2021 BIDS-MATLAB developers

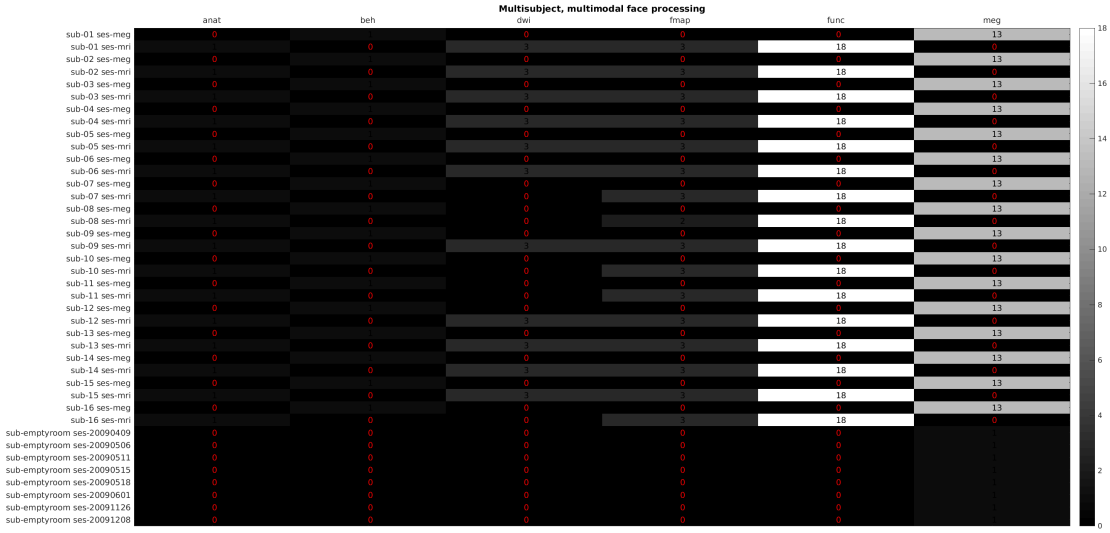

Fig. 1: output of `diagsnotic`

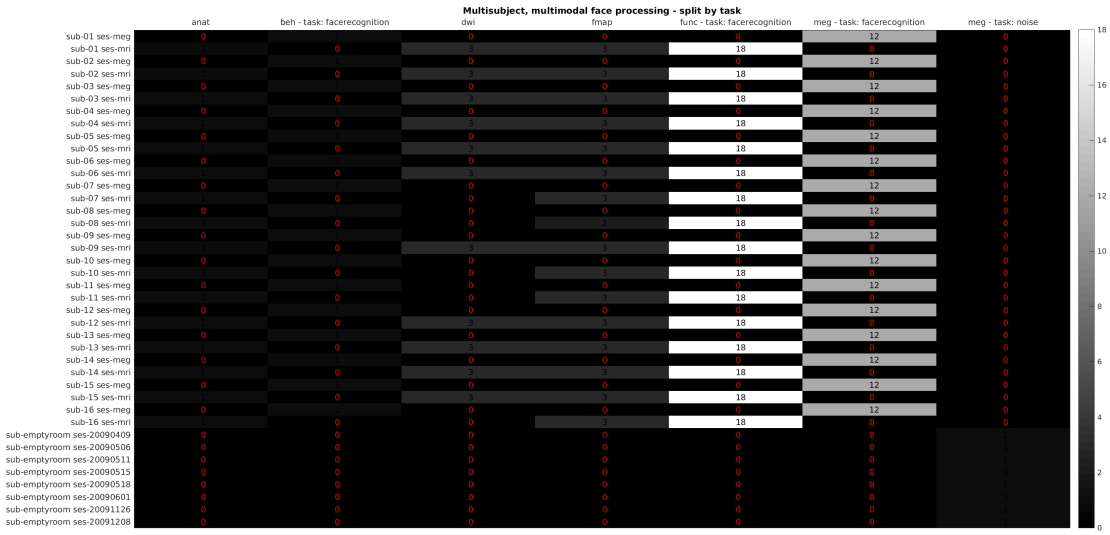

Fig. 2: output of `diagsnotic` split by task

CHAPTER

FOUR

UTILITY FUNCTIONS

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

Decode JSON-formatted file

USAGE:

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

Parameters

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

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

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

Returns

- **json** JSON structure

(C) Copyright 2018 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

(C) Copyright 2018 BIDS-MATLAB developers

+bids.+util.**jsonwrite**(*varargin*)

Serialize a JSON (JavaScript Object Notation) structure

USAGE:

```
jsonwrite(filename, json)
```

Parameters

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

USAGE:

```
S = jsonwrite(json)
```

Parameters **json** (structure) – JSON structure

Returns S - serialized JSON structure (string)

USAGE:

```
[...] = jsonwrite(..., opts)
```

Parameters **opts** (structure or list) – name/value pairs of optional parameters.

- 'prettyPrint': indent output [Default: true]
- 'replacementStyle': string to control how non-alphanumeric characters are replaced {'underscore', 'hex', 'delete', 'nop'} [Default: 'underscore']
- 'convertInfAndNaN': encode NaN, Inf and -Inf as "null" [Default: true]

References:

- JSON Standard
- jsonencode

(C) Copyright 2018 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

$Id: spm_jsonwrite.m 8031 2020-12-10 13:37:00Z guillaume $

+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** (string) – filename (can be gzipped) {txt,mat,csv,tsv,json}ename
- **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** (boolean) – 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.

(C) Copyright 2018 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

(C) Copyright 2018 BIDS-MATLAB developers

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

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

USAGE:

```
tsvwrite(f, var)
```

Parameters

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

Based on spm_save.m from SPM12.

(C) Copyright 2018 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

(C) Copyright 2018 BIDS-MATLAB developers

+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.

EXAMPLE:

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

Based on spm_mkdir from SPM12

(C) Copyright 2017-2021 Guillaume Flandin, Wellcome Centre for Human Neuroimaging

(C) Copyright 2018 BIDS-MATLAB developers

CHAPTER

FIVE

USING THE BIDS SCHEMA

class +bids.**Schema**(*use_schema*)

Class to interact with the BIDS schema

USAGE:

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

use_schema: boolean

(C) Copyright 2021 BIDS-MATLAB developers

Schema(*use_schema*)

Constructor

load(*use_schema*)

Loads a json schema by recursively looking through a folder structure.

The nesting of the output structure reflects a combination of the folder structure and any eventual nesting within each json.

USAGE:

```
schema = bids.Schema
schema = schema.load
```

return_modalities(*subject*, *modality_group*)

if we go schema-less or use another schema than the "official" one we list directories in the subject/session folder as proxy of the modalities that we have to parse

return_modality_groups()

Returns a dummy variable if we go schema less

required_entities_for_suffix_group(*this_suffix_group*)

Returns 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)
```

find_suffix_group(*modality*, *suffix*)

For a given sufffix and modality, this returns the "suffix group" this suffix belongs to

USAGE:

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

return_datatypes_for_suffix(*suffix*)
For a given suffix, returns all the possible datatypes that have this suffix.

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)
```

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)
```

return_modality_extensions_regex(*modality*)
creates a regular expression of extensions for a given imaging modality

USAGE:

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

return_modality_regex(*modality*)
creates a regular expression of suffixes and extension for a given imaging modality

USAGE:

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

static append_json_to_schema(*structure*, *json_file_list*)
Reads a json file and appends its content to the bids schema

USAGE:

```
structure = append_json_to_schema(structure, json_file_list)
```

static inspect_subdir(*structure*, *subdir_list*)
Recursively inspects subdirectory for json files and reflects folder hierarchy in the output structure.

USAGE:

```
structure = inspect_subdir(obj, structure, subdir_list)
```

static ci_check(*variable_to_check*)
Mostly to avoid some crash in continuous integration

CHAPTER

SIX

INDICES AND TABLES

- genindex

MATLAB MODULE INDEX

INDEX

Symbols

A

B

C

D

E

F

G

I

J

L

M

P

Q

R

S

T

U

V

W