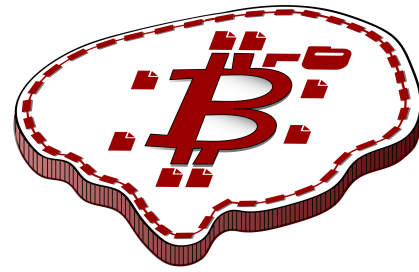

BIDScoin

Release 3.0.5

Feb 04, 2021

1	BIDScoin functionality	3
2	Note:	5
2.1	Installation	5
2.1.1	Dcm2niix installation	5
2.1.2	Python 3 installation	5
2.1.3	BIDScoin installation	5
2.2	Data preparation	6
2.2.1	Required source data structure	6
2.2.2	Data management utilities	8
2.3	The BIDScoin workflow	11
2.3.1	Step 1a: Running the bidsmapper	11
2.3.2	Step 1b: Running the bidseditor	12
2.3.3	Step 2: Running the bidscoiner	16
2.4	Finishing up	17
2.4.1	Adding meta-data	17
2.4.2	Data sharing utilities	17
2.4.3	BIDS validation	19
2.5	Options	19
2.5.1	BIDScoin	19
2.5.2	dcm2niix	19
2.5.3	Plugins	20
2.6	Advanced usage	20
2.6.1	Site specific / customized template	20
2.6.2	Plugins	23
2.7	Screenshots	24
2.7.1	The bidseditor	24
2.8	Demo and tutorial	24
2.8.1	BIDS introduction and BIDScoin demo	24
2.8.2	Hands-on tutorial	24



BIDScoin

BIDScoin is a user friendly [open-source](#) python toolkit that converts (“coins”) source-level (raw) neuroimaging data-sets to [nifti](#) / [json](#) / [tsv](#) data-sets that are organized following the Brain Imaging Data Structure, a.k.a. the [BIDS](#) standard. Rather than depending on complex or ambiguous programmatic logic for the identification of imaging modalities, BIDScoin uses a direct mapping approach to identify and convert the raw source data into BIDS data. Different runs of source data are identified by reading information from MRI header files (DICOM or PAR/REC; e.g. ‘ProtocolName’) and the mapping information about how these different runs should be named in BIDS can be specified a priori as well as interactively by the researcher – bringing in the missing knowledge that often exists only in his or her head!

Because all the mapping information can be easily edited with a Graphical User Interface (GUI), BIDScoin requires no programming knowledge in order to use it.

BIDScoin is developed at the [Donders Institute](#) of the [Radboud University](#).

CHAPTER 1

BIDScoin functionality

- ☒ DICOM source data
- ☒ PAR / REC source data (Philips)
- ☐ P7 source data (GE)
- ☐ Nifti source data
- ☒ Fieldmaps*
- ☒ Multi-echo data*
- ☒ Multi-coil data*
- ☒ PET data*
- ☐ Stimulus / behavioural logfiles
- ☐ Physiological data
- ☒ Plug-ins
- ☒ Defacing
- ☒ Multi-echo combination

* = Experimental support for PAR / REC source data

Are you a python programmer with an interest in BIDS who knows all about GE and / or ↪
↪Philips data?
Are you experienced with parsing stimulus presentation log-files? Or do you have ↪
↪ideas to improve
the this toolkit or its documentation? Have you come across bugs? Then you are highly ↪
↪encouraged to
provide feedback or contribute to this project on <https://github.com/Donders->
↪Institute/bidscoin.

Note:

The full BIDScoin documentation is hosted at [Read the Docs](#)

Issues can be reported at [Github](#)

2.1 Installation

BIDScoin can be installed and should work on Linux, MS Windows and on OS-X computers (this latter option has not been tested) that satisfy the system requirements:

- dcm2niix
- python 3.6 or higher

2.1.1 Dcm2niix installation

BIDScoin relies on dcm2niix to convert the source imaging data to nifti. Please download and install [dcm2niix](#) yourself according to the instructions. When done, make sure that the path to the dcm2niix binary / executable is set correctly in the BIDScoin Options in the `[path_to_bidscoin]/heuristics/bidsmap_template.yaml` file or in the [Site specific / customized template](#) file.

2.1.2 Python 3 installation

BIDScoin is a python package and therefore a python interpreter needs to be present on the system. On Linux this is usually already the case, but MS Windows users may need to install python themselves. See e.g. [this python 3 distribution](#) for instructions. They may also need to install the [MS Visual C++](#) build tools (sorry for this pain).

2.1.3 BIDScoin installation

To install BIDScoin run the following command in your command-shell (institute users may want to activate a [virtual / conda](#) python environment first):

```
$ pip install bidscoin
```

This will give you the latest stable release of the software. To get the very latest (development) version of the software you can install the package directly from the github source code repository:

```
$ pip install git+https://github.com/Donders-Institute/bidscoin
```

If you want to edit the code or want to contribute back to the project, you can use the `-e` option:

```
$ pip install -e git+https://github.com/Donders-Institute/bidscoin#egg=bidscoin
```

If you do not have git (or any other version control system) installed you can [download](#) and unzip the code yourself in a directory named e.g. `bidscoin` and run (again, with or without the `-e` option):

```
$ pip install -e bidscoin
```

Updating BIDScoin

Run the pip command as before with the additional `--upgrade` option:

```
$ pip install --upgrade bidscoin
```

Caution:

- The bidsmaps are not guaranteed to be compatible between different BIDScoin versions.
- After a succesful BIDScoin installation or upgrade, it may be needed to (re)do any adjustments that were done on the [Site specific / customized template](#) file(s) (so make a back-up of these before you upgrade).

2.2 Data preparation

2.2.1 Required source data structure

BIDScoin requires that the source data input folder is organized according to a `sub-identifier/[ses-identifier]/sessiondata` structure (NB: the `ses-identifier` subfolder is optional). The `sessiondata` can have various formats, as shown in the following examples:

1. **A ‘seriesfolder’ organization.** A series folder contains a single data type and are typically acquired in a single run – a.k.a ‘Series’ in DICOM speak. This is how users receive their data from the (Siemens) scanners at the DCCN:

```
sourcedata
|-- sub-001
|   |-- ses-mri01
|   |   |-- 001-localizer
|   |   |   |-- 00001_1.3.12.2.1107.5.2.19.45416.2017121914582956872274162.IMA
|   |   |   |-- 00002_1.3.12.2.1107.5.2.19.45416.2017121914583757650874172.IMA
|   |   |   |-- 00003_1.3.12.2.1107.5.2.19.45416.2017121914583358068374167.IMA
|   |   |
|   |   |-- 002-t1_mprage_sag_p2_iso_1.0
|   |   |   |-- 00002_1.3.12.2.1107.5.2.19.45416.2017121915051526005675150.IMA
```

(continues on next page)

(continued from previous page)

```

| | | |-- 00003_1.3.12.2.1107.5.2.19.45416.2017121915051520026075138.IMA
| | | |-- 00004_1.3.12.2.1107.5.2.19.45416.2017121915051515689275130.IMA
| | | [...]
| | | [...]
| | |
| | `-- ses-mri02
| |     |-- 001-localizer
| |     | |-- 00001_1.3.12.2.1107.5.2.19.45416.2017121914582956872274162.IMA
| |     | |-- 00002_1.3.12.2.1107.5.2.19.45416.2017121914583757650874172.IMA
| |     | `-- 00003_1.3.12.2.1107.5.2.19.45416.2017121914583358068374167.IMA
| |     [...]
| |
|-- sub-002
| | `-- ses-mri01
| |     |-- 001-localizer
| |     | |-- 00001_1.3.12.2.1107.5.2.19.45416.2017121914582956872274162.IMA
| |     | |-- 00002_1.3.12.2.1107.5.2.19.45416.2017121914583757650874172.IMA
| |     | `-- 00003_1.3.12.2.1107.5.2.19.45416.2017121914583358068374167.IMA
| |     [...]
| |
[...]
```

2. **A ‘DICOMDIR’ organization.** A DICOMDIR is dictionary-file that indicates the various places where all the DICOM files are stored of each DICOM Series. This is how data is often exported in clinical settings:

```

sourcedata
|-- sub-001
| | |-- DICOM
| | | `-- 00001EE9
| | |     |-- AAFC99B8
| | |     |-- AA547EAB
| | |         |-- 00000025
| | |         | |-- EE008C45
| | |         | |-- EE027F55
| | |         | |-- EE03D17C
| | |         | [...]
| | |         |
| | |         |-- 000000B4
| | |         | |-- EE07CCDA
| | |         | |-- EE0E0701
| | |         | |-- EE0E200A
| | |         | [...]
| | |         [...]
| | `-- DICOMDIR
| |
|-- sub-002
| | [...]
| |
[...]
```

3. **A flat DICOM organization.** In a flat DICOM organization all the DICOM files of all the different Series are simply put in one large directory. This is how data is sometimes exported in clinical settings:

```

sourcedata
|-- sub-001
| | `-- ses-mri01
| |     |-- IM_0001.dcm
| |     |-- IM_0002.dcm
```

(continues on next page)

(continued from previous page)

```
|      |-- IM_0003.dcm
|      [...]
|
|-- sub-002
|   |-- ses-mri01
|       |-- IM_0001.dcm
|       |-- IM_0002.dcm
|       |-- IM_0003.dcm
|       [...]
|   [...]
[...]
```

4. **A PAR/REC organization.** All PAR/REC(/XML) files of all the different Series are put in one directory. This is how users often export their data from Philips scanners in research settings:

```
sourcedata
|-- sub-001
|   |-- ses-mri01
|       |-- TCHC_066_1_WIP_Hanneke_Block_2_SENSE_4_1.PAR
|       |-- TCHC_066_1_WIP_Hanneke_Block_2_SENSE_4_1.REC
|       |-- TCHC_066_1_WIP_IDED_SENSE_6_1.PAR
|       |-- TCHC_066_1_WIP_IDED_SENSE_6_1.REC
|       |-- TCHC_066_1_WIP_Localizer_CLEAR_1_1.PAR
|       |-- TCHC_066_1_WIP_Localizer_CLEAR_1_1.REC
|       [...]
|   [...]
|-- sub-002
|   |-- ses-mri01
|       |-- TCHC_066_1_WIP_Hanneke_Block_2_SENSE_4_1.PAR
|       |-- TCHC_066_1_WIP_Hanneke_Block_2_SENSE_4_1.REC
|       |-- TCHC_066_1_WIP_IDED_SENSE_6_1.PAR
|       |-- TCHC_066_1_WIP_IDED_SENSE_6_1.REC
|       |-- TCHC_066_1_WIP_Localizer_CLEAR_1_1.PAR
|       |-- TCHC_066_1_WIP_Localizer_CLEAR_1_1.REC
|       [...]
|   [...]
[...]
```

Note: You can store the `sessiondata` in any of the above data organizations as zipped (`.zip`) or tarzipped (e.g. `.tar.gz`) archive files. BIDScoin [workflow tools](#) will unpack/unzip those archive files in a temporary folder and will process the `sessiondata` from there. The BIDScoin tools will run `dicomsort` in a temporary folder for flat/DICOMDIR data to sort them in seriesfolders. BIDScoin tools that work from a temporary folder has the downside of getting a speed penalty. Also note that privacy-sensitive data samples will then be stored in `[bidsfolder]/code/bidscoin/provenance`.

2.2.2 Data management utilities

dicomsort

The `dicomsort` command-line tool is a utility to move your flat- or DICOMDIR-organized files (see [above](#)) into a ‘seriesfolder’ organization (see above). This can be useful to organise your source data in a more convenient and human readable way, as DICOMDIR or flat DICOM directories can often be hard to comprehend. The BIDScoin tools will run `dicomsort` in a temporary folder if your data is not already organised in series-folders, so in principle you don’t really need to run it yourself. Running `dicomsort` beforehand does, however, give you more flexibility in handling special cases that are not handled properly and it can also give you a speed benefit.

```
usage: dicomsort [-h] [-i SUBPREFIX] [-j SESPREFIX] [-f FIELDNAME] [-r]
               [-e EXT] [-n] [-p PATTERN] [-d]
               dicomsource

Sorts and / or renames DICOM files into local subdirectories with a (3-digit)
SeriesNumber-SeriesDescription directory name (i.e. following the same listing
as on the scanner console)

positional arguments:
  dicomsource           The name of the root folder containing the
                        dicomsource/[sub/][ses/]dicomfiles and / or the
                        (single session/study) DICOMDIR file

optional arguments:
  -h, --help            show this help message and exit
  -i SUBPREFIX, --subprefix SUBPREFIX
                        Provide a prefix string for recursive searching in
                        dicomsource/subject subfolders (e.g. "sub") (default:
                        None)
  -j SESPREFIX, --sesprefix SESPREFIX
                        Provide a prefix string for recursive searching in
                        dicomsource/subject/session subfolders (e.g. "ses")
                        (default: None)
  -f FIELDNAME, --fieldname FIELDNAME
                        The dicomfield that is used to construct the series
                        folder name ("SeriesDescription" and "ProtocolName"
                        are both used as fallback) (default:
                        SeriesDescription)
  -r, --rename           Flag to rename the DICOM files to a PatientName_Series
                        Number_SeriesDescription_AcquisitionNumber_InstanceNum
                        ber scheme (recommended for DICOMDIR data) (default:
                        False)
  -e EXT, --ext EXT      The file extension after sorting (empty value keeps
                        the original file extension), e.g. ".dcm" (default: )
  -n, --nosort           Flag to skip sorting of DICOM files into SeriesNumber-
                        SeriesDescription directories (useful in combination
                        with -r for renaming only) (default: False)
  -p PATTERN, --pattern PATTERN
                        The regular expression pattern used in
                        re.match(pattern, dicomfile) to select the dicom files
                        (default: .*\. (IMA|dcm)$)
  -d, --dryrun           Add this flag to just print the dicomsort commands
                        without actually doing anything (default: False)

examples:
dicomsort /project/3022026.01/raw
dicomsort /project/3022026.01/raw --subprefix sub
dicomsort /project/3022026.01/raw --subprefix sub-01 --sesprefix ses
dicomsort /project/3022026.01/raw/sub-011/ses-mri01/DICOMDIR -r -e .dcm
```

rawmapper

Another command-line utility that can be helpful in organizing your source data is `rawmapper`. This utility can show you the overview (map) of all the values of DICOM-fields of interest in your data-set and, optionally, use these fields to rename your source data sub-folders (this can be handy e.g. if you manually entered subject-identifiers as [Additional info] at the scanner console and you want to use these to rename your subject folders).

```
usage: rawmapper [-h] [-s SESSIONS [SESSIONS ...]]
                [-d DICOMFIELD [DICOMFIELD ...]] [-w WILDCARD]
                [-o OUTFOLDER] [-r] [-n SUBPREFIX] [-m SESPREFIX]
                [--dryrun]
                sourcefolder
```

Maps out the values of a dicom field of all subjects in the sourcefolder, saves the result in a mapper-file and, optionally, uses the dicom values to rename the sub-/ses-id's of the subfolders. This latter option can be used, e.g. when an alternative subject id was entered in the [Additional info] field during subject registration (i.e. stored in the PatientComments dicom field)

positional arguments:

sourcefolder	The source folder with the raw data in sub-#/ses-#/series organisation
--------------	--

optional arguments:

-h, --help	show this help message and exit
-s SESSIONS [SESSIONS ...], --sessions SESSIONS [SESSIONS ...]	Space separated list of selected sub-#/ses-# names / folders to be processed. Otherwise all sessions in the bidsfolder will be selected (default: None)
-d DICOMFIELD [DICOMFIELD ...], --dicomfield DICOMFIELD [DICOMFIELD ...]	The name of the dicomfield that is mapped / used to rename the subid/sesid foldernames (default: ['PatientComments'])
-w WILDCARD, --wildcard WILDCARD	The Unix style pathname pattern expansion that is used to select the series from which the dicomfield is being mapped (can contain wildcards) (default: *)
-o OUTFOLDER, --outfolder OUTFOLDER	The mapper-file is normally saved in sourcefolder or, when using this option, in outfolder (default: None)
-r, --rename	If this flag is given sub-subid/ses-sesid directories in the sourcefolder will be renamed to sub-dcmval/ses-dcmval (default: False)
-n SUBPREFIX, --subprefix SUBPREFIX	The prefix common for all the source subject-folders (default: sub-)
-m SESPREFIX, --sesprefix SESPREFIX	The prefix common for all the source session-folders (default: ses-)
--dryrun	Add this flag to dryrun (test) the mapping or renaming of the sub-subid/ses-sesid directories (i.e. nothing is stored on disk and directory names are not actually changed) (default: False)

examples:

```
rawmapper /project/3022026.01/raw/
rawmapper /project/3022026.01/raw -d AcquisitionDate
rawmapper /project/3022026.01/raw -s sub-100/ses-mri01 sub-126/ses-mri01
rawmapper /project/3022026.01/raw -r -d ManufacturerModelName AcquisitionDate --
↪dryrun
rawmapper raw/ -r -s sub-1*/* sub-2*/ses-mri01 --dryrun
rawmapper -d EchoTime -w *fMRI* /project/3022026.01/raw
```

Note: If these data management utilities do not satisfy your needs, then have a look at this [reorganize_dicom_files](#)

tool.

2.3 The BIDScoin workflow

Having an organized source data folder, the actual data-set conversion to BIDS is performed by the *(1a)* the `bidsmapper`, *(1b)* the `bidseditor` and *(2)* the `bidscoiner` command-line tools. The `bidsmapper` makes a map of the different kind of datatypes in your source dataset, with the `bidseditor` you can edit this map, and the `bidscoiner` does the actual work to convert the source data into BIDS. By default (but see the `-i` option of the `bidsmapper` below), step 1a automatically launches step 1b, so in it's simplest form, all you need to do to convert your raw source data into BIDS is to run two simple commands, e.g.:

```
$ bidsmapper sourcefolder bidsfolder
$ bidscoiner sourcefolder bidsfolder
```

2.3.1 Step 1a: Running the bidsmapper

```
usage: bidsmapper [-h] [-b BIDSMAP] [-t TEMPLATE] [-n SUBPREFIX]
                 [-m SESPREFIX] [-i {0,1,2}] [-v]
                 sourcefolder bidsfolder
```

Creates a `bidsmap.yaml` YAML file in the `bidsfolder/code/bidscoin` that maps the ↵
 ↵information
 from all raw source data to the BIDS labels. You can check and edit the `bidsmap` file ↵
 ↵with
 the `bidseditor` (but also with any text-editor) before passing it to the `bidscoiner`. ↵
 ↵See the
`bidseditor` help for more information and useful tips for running the `bidsmapper` in ↵
 ↵interactive
 mode (which is the default).

N.B.: Institute users may want to use a site-customized template `bidsmap` (see the `--template` option). The `bidsmap_dcn` template from the Donders Institute can serve as an example (or may even mostly work for other institutes out of the box).

positional arguments:

<code>sourcefolder</code>	The study root folder containing the raw data in sub-#[ses-#]/data subfolders (or specify <code>--subprefix</code> and <code>--sesprefix</code> for different prefixes)
<code>bidsfolder</code>	The destination folder with the (future) bids data and the <code>bidsfolder/code/bidscoin/bidsmap.yaml</code> output file

optional arguments:

<code>-h, --help</code>	show this help message and exit
<code>-b BIDSMAP, --bidsmap BIDSMAP</code>	The <code>bidsmap</code> YAML-file with the study heuristics. If the <code>bidsmap</code> filename is relative (i.e. no "/" in the name) then it is assumed to be located in <code>bidsfolder/code/bidscoin</code> . Default: <code>bidsmap.yaml</code>
<code>-t TEMPLATE, --template TEMPLATE</code>	The <code>bidsmap</code> template with the default heuristics (this could be provided by your institute). If the <code>bidsmap</code> filename is relative (i.e. no "/" in the name) then it

(continues on next page)

(continued from previous page)

```

        is assumed to be located in bidsfolder/code/bidscoin.
        Default: bidsmap_template.yaml
-n SUBPREFIX, --subprefix SUBPREFIX
        The prefix common for all the source subject-folders.
        Default: 'sub-'
-m SESPREFIX, --sesprefix SESPREFIX
        The prefix common for all the source session-folders.
        Default: 'ses-'
-s, --store
        Flag to store the provenance data samples in the
        bidsfolder/'code'/'provenance' folder
-i {0,1,2}, --interactive {0,1,2}
        {0}: The sourcefolder is scanned for different kinds
        of scans without any user interaction. {1}: The
        sourcefolder is scanned for different kinds of scans
        and, when finished, the resulting bidsmap is opened
        using the bidseditor. {2}: As {1}, except that already
        during scanning the user is asked for help if a new
        and unknown run is encountered. This option is most
        useful when re-running the bidsmapper (e.g. when the
        scan protocol was changed since last running the
        bidsmapper). Default: 1
-v, --version
        Show the BIDS and BIDScoin version

examples:
  bidsmapper /project/foo/raw /project/foo/bids
  bidsmapper /project/foo/raw /project/foo/bids -t bidsmap_dccn

```

The bidsmapper will scan your sourcefolder to look for different runs (scan-types) to create a mapping for each run to a bids output name (a.k.a. the ‘bidsmap’). By default (but see the `-i` option above), when finished the bidsmapper will automatically launch [step 1b](#), as described in the next section (but step 1b can also always be run separately by directly running the bidseditor).

Tip: Use the `-t bidsmap_dccn` option and see if it works for you. If not, consider [adapting it to your needs](#).

2.3.2 Step 1b: Running the bidseditor

```

usage: bidseditor [-h] [-b BIDSMAP] [-t TEMPLATE] [-d DATAFORMAT]
                [-n SUBPREFIX] [-m SESPREFIX]
                bidsfolder

```

This tool launches a graphical user interface for editing the bidsmap.yaml file that is e.g. produced by the bidsmapper or by this bidseditor itself. The user can fill in or change the BIDS labels for entries that are unidentified or sub-optimal, such that meaningful BIDS output names will be generated from these labels. The saved bidsmap.yaml output file can be used for converting the source data to BIDS using the bidscoiner.

positional arguments:

```

  bidsfolder      The destination folder with the (future) bids data

```

optional arguments:

```

  -h, --help      show this help message and exit
  -b BIDSMAP, --bidsmap BIDSMAP

```

(continues on next page)

(continued from previous page)

```

    The bidsmap YAML-file with the study heuristics. If
    the bidsmap filename is relative (i.e. no "/" in the
    name) then it is assumed to be located in
    bidsfolder/code/bidscoin. Default: bidsmap.yaml
-t TEMPLATE, --template TEMPLATE
    The bidsmap template with the default heuristics (this
    could be provided by your institute). If the bidsmap
    filename is relative (i.e. no "/" in the name) then it
    is assumed to be located in bidsfolder/code/bidscoin.
    Default: bidsmap_template.yaml
-d DATAFORMAT, --dataformat DATAFORMAT
    The format of the source data, e.g. DICOM or PAR.
    Default: DICOM
-n SUBPREFIX, --subprefix SUBPREFIX
    The prefix common for all the source subject-folders.
    Default: 'sub-'
-m SESPREFIX, --sesprefix SESPREFIX
    The prefix common for all the source session-folders.
    Default: 'ses-'

```

examples:

```

bidseditor /project/foo/bids
bidseditor /project/foo/bids -t bidsmap_dccn.yaml
bidseditor /project/foo/bids -b my/custom/bidsmap.yaml

```

Here are a few tips & tricks:

DICOM Attributes

An (DICOM) attribute label can also be a list, in which case the BIDS labels /
↳ mapping
are applied if a (DICOM) attribute value is in this list. If the attribute value is
empty it is not used to identify the run. Wildcards can also be given, either as a
↳ single
'*, or enclosed by '*'. Examples:

```

SequenceName: '*'
SequenceName: '*epfid*'
SequenceName: ['epfid2d1rs', 'fm2d2r']
SequenceName: ['*epfid*', 'fm2d2r']

```

NB: Editing the DICOM attributes is normally not necessary and advised against

Dynamic BIDS labels

The BIDS labels can be static, in which case the label is just a normal string, or
↳ dynamic,
when the string is enclosed with pointy brackets like `` or
`<<argument1><argument2>>`. In case of single pointy brackets the label will be
↳ replaced
during bidsmapper, bidseditor and bidscoiner runtime by the value of the (DICOM)
↳ attribute
with that name. In case of double pointy brackets, the label will be updated for
↳ each
subject/session during bidscoiner runtime. For instance, then the `run` label `<<1>>`
↳ ` in
the bids name will be replaced with `1` or increased to `2` if a file with runindex
↳ `1`
already exists in that directory.

(continues on next page)

(continued from previous page)

```

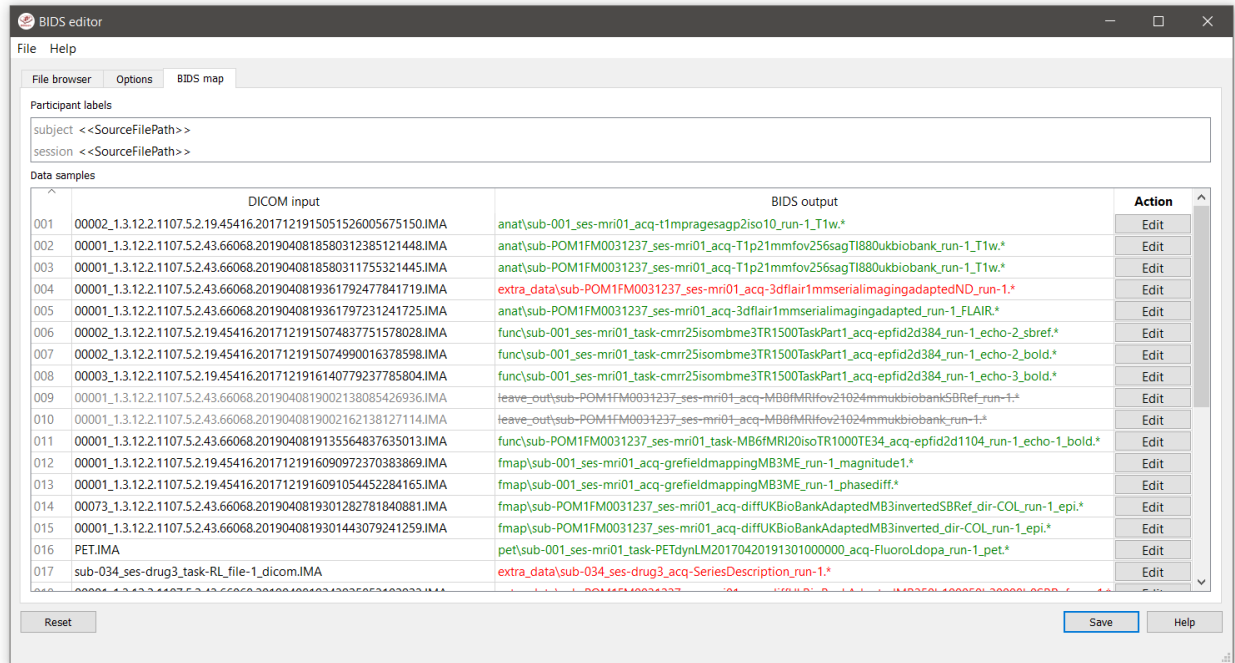
Fieldmaps: suffix
  Select 'magnitude1' if you have 'magnitude1' and 'magnitude2' data in one series-
  ↳ folder
  (this is what Siemens does) -- the bidscoiner will automatically pick up the
  ↳ 'magnitude2'
  data during runtime. The same holds for 'phase1' and 'phase2' data. See the BIDS
  specification for more details on fieldmap suffixes

Fieldmaps: IntendedFor
  You can use the `IntendedFor` field to indicate for which runs (DICOM series) a
  ↳ fieldmap
  was intended. The dynamic label of the `IntendedFor` field can be a list of string
  ↳ patterns
  that is used to include all runs in a session that have that string pattern in
  ↳ their BIDS
  file name. Example: use `<<task>>` to include all functional runs or `<<Stop*Go>
  ↳ <Reward>>`
  to include "Stop1Go"-, "Stop2Go"- and "Reward"-runs.
  NB: The fieldmap might not be used at all if this field is left empty!

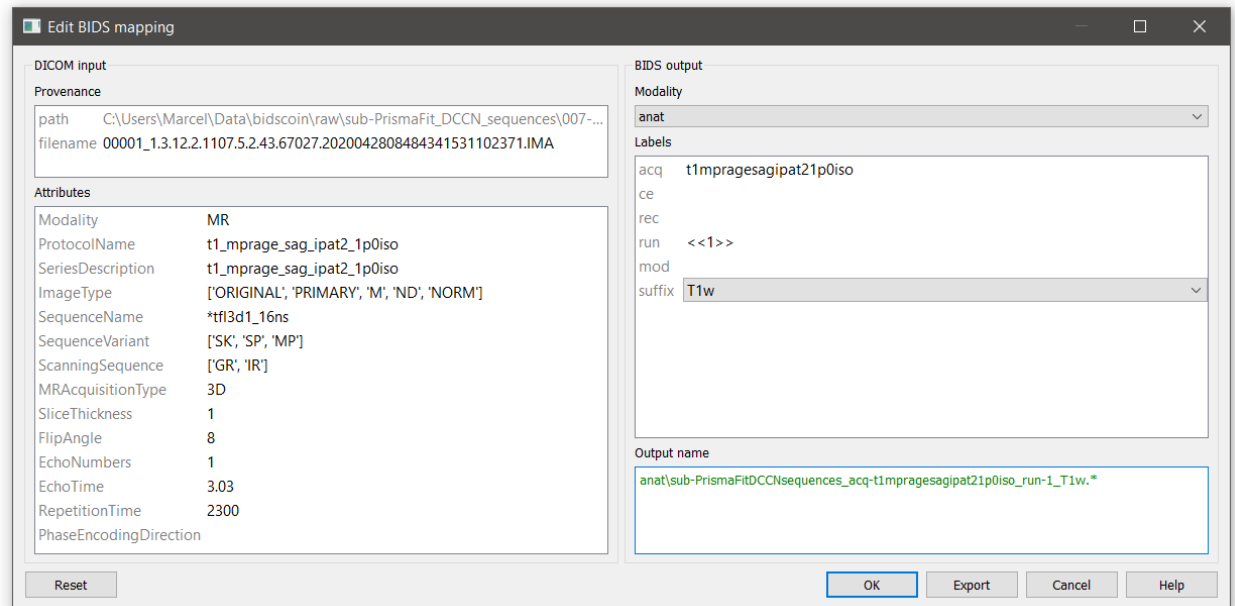
Manual editing / inspection of the bidsmap
  You `can of course also directly edit or inspect the `bidsmap.yaml` file yourself
  ↳ with any
  text editor. For instance to merge a set of runs that by adding a '*' wildcard to a
  ↳ DICOM
  attribute in one run item and then remove the other runs in the set. See ./docs/
  ↳ bidsmap.md
  and ./heuristics/bidsmap_dccn.yaml for more information.

```

As shown below, the main window of the bidseditor opens with the BIDS map tab that contains a list of input samples that uniquely represents all the different files that are present in the source folder, together with the associated BIDS output name. The path in the BIDS output name is shown in red if the modality is not part of the BIDS standard, striked-out gray when the runs will be ignored in the conversion to BIDS, otherwise it is colored green. Double clicking the sample (DICOM) filename opens an inspection window with the full header information (double clicking sample filenames works throughout the GUI).



The user can click the Edit button for each list item to open a new edit window, as shown below. In this interface, the right BIDS Modality (drop down menu) and the suffix label (drop down menu) can be set correctly, after which the associated BIDS Labels can be edited (double click black items). As a result, the new BIDS Output name is then shown in the bottom text field. This is how the BIDS output data will look like and, if this looks all fine, the user can store this mapping to the bidsmap and return to the main window by clicking the OK button.



Tip: Use the Export button to append new or unknown run items to your template bidsmap for usage in other studies

Finally, if all BIDS output names in the main window are fine, the user can click on the Save button and proceed with

running the bidscoiner tool.

2.3.3 Step 2: Running the bidscoiner

```
usage: bidscoiner [-h] [-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]] [-f]
                  [-s] [-b BIDSMAP] [-n SUBPREFIX] [-m SESPREFIX] [-v]
                  sourcefolder bidsfolder
```

Converts ("coins") datasets in the sourcefolder to nifti / json / tsv datasets in the bidsfolder according to the BIDS standard. Check and edit the bidsmap.yaml file to your needs using the bidseditor tool before running this function. You can run bidscoiner after all data is collected, or run / re-run it whenever new data has been added to the source folder (presuming the scan protocol hasn't changed). If you delete a (subject/) session folder from the bidsfolder, it will be re-created from the sourcefolder the next time you run the bidscoiner.

Provenance information, warnings and error messages are stored in the bidsfolder/code/bidscoin/bidscoiner.log file.

positional arguments:

sourcefolder	The source folder containing the raw data in sub-#[/ses-#]/data format (or specify --subprefix and --sesprefix for different prefixes)
bidsfolder	The destination / output folder with the bids data

optional arguments:

-h, --help	show this help message and exit
-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...], --participant_label PARTICIPANT_LABEL ↵ ↵[PARTICIPANT_LABEL ...]	Space separated list of selected sub-# names / folders to be processed (the sub- prefix can be removed). Otherwise all subjects in the sourcefolder will be selected
-f, --force	If this flag is given subjects will be processed, regardless of existing folders in the bidsfolder. Otherwise existing folders will be skipped
-s, --skip_participants	If this flag is given those subjects that are in participants.tsv will not be processed (also when the --force flag is given). Otherwise the participants.tsv table is ignored
-b BIDSMAP, --bidsmap BIDSMAP	The bidsmap YAML-file with the study heuristics. If the bidsmap filename is relative (i.e. no "/" in the name) then it is assumed to be located in bidsfolder/code/bidscoin. Default: bidsmap.yaml
-n SUBPREFIX, --subprefix SUBPREFIX	The prefix common for all the source subject-folders. Default: 'sub-'
-m SESPREFIX, --sesprefix SESPREFIX	The prefix common for all the source session-folders. Default: 'ses-'
-v, --version	Show the BIDS and BIDScoin version

examples:

```
bidscoiner /project/foo/raw /project/foo/bids
bidscoiner -f /project/foo/raw /project/foo/bids -p sub-009 sub-030
```

Tip: Check your json sidecar files of your fieldmaps, in particular see if they have the expected `IntendedFor` values.

Note: The provenance of the produced BIDS data-sets is stored in the `[bidsfolder]/code/bidscoin/bidscoiner.log` file. This file is also very useful for debugging / tracking down bidscoin issues.

2.4 Finishing up

After a successful run of `bidscoiner`, the work to convert your data in a fully compliant BIDS dataset is unfortunately not yet fully over and, depending on the complexity of your data-set, additional tools may need to be run and meta-data may need to be entered manually (not everything can be automated).

2.4.1 Adding meta-data

For instance, you should update the content of the `dataset_description.json` and `README` files in your bids folder and you may need to provide e.g. additional `*_scans.tsv`, `*_sessions.tsv` or `participants.json` files (see the [BIDS specification](#) for more information). Moreover, if you have behavioural log-files you will find that BIDScoin does not (yet) [support](#) converting these into BIDS compliant `*_events.tsv/json` files (advanced users are encouraged to use the `bidscoiner` [plug-in](#) possibility and write their own log-file parser).

2.4.2 Data sharing utilities

Multi-echo combination

Before sharing or pre-processing their images, users may want to combine the separate the individual echos of multi-echo MRI acquisitions. The `echcombine`-tool is a wrapper around `mecombine` that writes BIDS valid output.

```
usage: echcombine [-h] [-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]
                  [-o {fmap,anat,func,dwi,beh,pet,extra_data,derivatives}]
                  [-a {PAID,TE,average}] [-w [WEIGHTS [WEIGHTS ...]]]
                  bidsfolder pattern
```

A wrapper around the 'mecombine' multi-echo combination tool (<https://github.com/Donders-Institute/multiecho>).

This wrapper is fully BIDS-aware (a 'bidsapp') and writes BIDS compliant output

positional arguments:

<code>bidsfolder</code>	The bids-directory with the (multi-echo) subject data
<code>pattern</code>	Globlike recursive search pattern (relative to the subject/session folder) to select the first echo of the images that need to be combined, e.g. <code>'*task-*echo-1*'</code>

optional arguments:

<code>-h, --help</code>	show this help message and exit
<code>-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...], --participant_label PARTICIPANT_LABEL</code>	
<code>↳ [PARTICIPANT_LABEL ...]</code>	

(continues on next page)

(continued from previous page)

```

        Space separated list of sub-# identifiers to be
        processed (the sub- prefix can be left out). If not
        specified then all sub-folders in the bidsfolder will
        be processed (default: None)
-o {fmap,anat,func,dwi,beh,pet,extra_data,derivatives}, --output {fmap,anat,func,
→dwi,beh,pet,extra_data,derivatives}
        A string that determines where the output is saved. It
        can be the name of a BIDS modality folder, such as
        'func', or of the derivatives folder, i.e.
        'derivatives'. If output = [the name of the input
        modality folder] then the original echo images are
        replaced by one combined image. If output is left
        empty then the combined image is saved in the input
        modality folder and the original echo images are moved
        to the extra_data folder (default: None)
-a {PAID,TE,average}, --algorithm {PAID,TE,average}
        Combination algorithm (default: TE)
-w [WEIGHTS [WEIGHTS ...]], --weights [WEIGHTS [WEIGHTS ...]]
        Weights for each echo (default: None)

examples:
echocombine /project/3017065.01/bids func/*task-stroop*echo-1*
echocombine /project/3017065.01/bids *task-stroop*echo-1* -p 001 003
echocombine /project/3017065.01/bids func/*task-*echo-1* -o func
echocombine /project/3017065.01/bids func/*task-*echo-1* -o derivatives -w 13 26 39
→52
echocombine /project/3017065.01/bids func/*task-*echo-1* -a PAID

```

Defacing

Before sharing or pre-processing their images, users may want to deface their anatomical MRI acquisitions as to protect the privacy of their subjects. The deface-tool is a wrapper around pydeface that writes BIDS valid output.

```

usage: deface [-h] [-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]]
             [-o {fmap,anat,func,dwi,beh,pet,extra_data,derivatives}] [-c]
             [-n NATIVESPEC] [-a ARGS]
             bidsfolder pattern

A wrapper around the 'pydeface' defacing tool (https://github.com/poldracklab/pydeface).
→pydeface).

This wrapper is fully BIDS-aware (a 'bidsapp') and writes BIDS compliant output

positional arguments:
  bidsfolder          The bids-directory with the (multi-echo) subject data
  pattern             Globlike search pattern (relative to the
                     subject/session folder) to select the images that need
                     to be defaced, e.g. 'anat/*_T1w*'

optional arguments:
  -h, --help          show this help message and exit
  -p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...], --participant_label PARTICIPANT_LABEL
→[PARTICIPANT_LABEL ...]
                     Space separated list of sub-# identifiers to be
                     processed (the sub- prefix can be left out). If not

```

(continues on next page)

(continued from previous page)

```

        specified then all sub-folders in the bidsfolder will
        be processed (default: None)
-o {fmap,anat,func,dwi,beh,pet,extra_data,derivatives}, --output {fmap,anat,func,
→dwi,beh,pet,extra_data,derivatives}
        A string that determines where the defaced images are
        saved. It can be the name of a BIDS modality folder,
        such as 'anat', or of the derivatives folder, i.e.
        'derivatives'. If output is left empty then the
        original images are replaced by the defaced images
        (default: None)
-c, --cluster
        Flag to submit the deface jobs to the high-performance
        compute (HPC) cluster (default: False)
-n NATIVESPEC, --nativespec NATIVESPEC
        DRMAA native specifications for submitting deface jobs
        to the HPC cluster (default: -l
        walltime=00:30:00,mem=1gb)
-a ARGS, --args ARGS
        Additional arguments (in dict/json-style) that are
        passed to pydeface. See examples for usage (default:
        {})

examples:
deface /project/3017065.01/bids anat/*_T1w*
deface /project/3017065.01/bids anat/*_T1w* -p 001 003 -o derivatives
deface /project/3017065.01/bids anat/*_T1w* -n "-l walltime=00:60:00,mem=2gb"
deface /project/3017065.01/bids anat/*_T1w* -a '{"cost": "corratio", "verbose": ""}'

```

2.4.3 BIDS validation

If all of the above work is done, you can (and should) run the web-based [bidsvalidator](#) to check for inconsistencies or missing files in your bids data-set (NB: the bidsvalidator also exists as a [command-line tool](#)).

2.5 Options

BIDScoin has different options and settings (see below) that can be adjusted per study bidsmap. You can use a text editor to edit the bidsmap template `[path_to_bidscoin]/heuristics/bidsmap_template.yaml` if you want to adjust the default)

2.5.1 BIDScoin

- `version`: should correspond with the version in `../bidscoin/version.txt`
- `bidsignore`: Semicolon-separated list of entries that are added to the `.bidsignore` file (for more info, see BIDS specifications), e.g.:
 - `extra_data/;pet;/myfile.txt;yourfile.csv`

2.5.2 dcm2niix

The nifti- and json-files are generated with [dcm2niix](#). Here you can adjust how dcm2niix is used:

- `path`: Command to set the path to dcm2niix, e.g.:

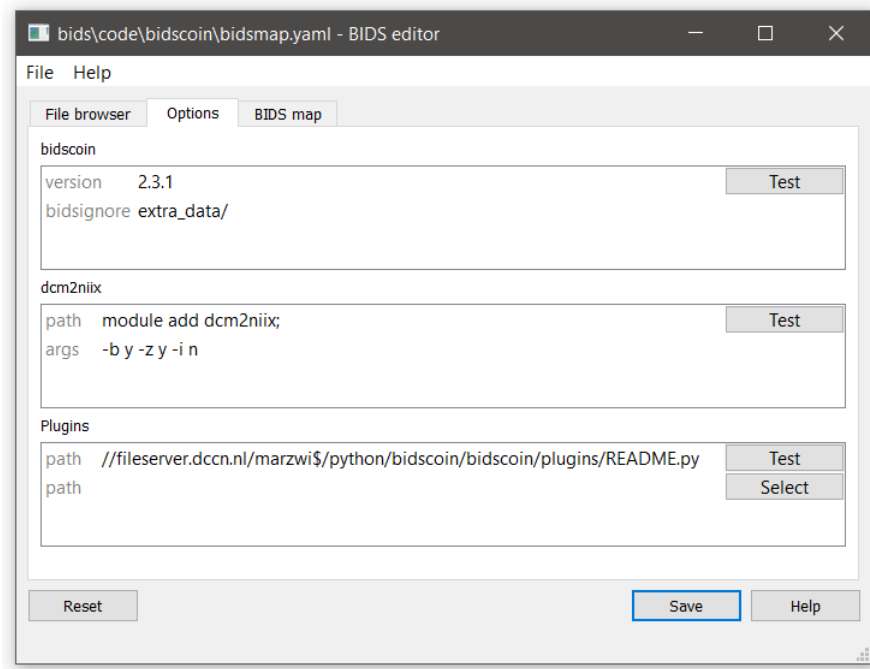


Fig. 1: The bidseditor options window with the different BIDScoin settings

- module add dcm2niix/1.0.20180622; (note the semi-colon at the end)
- PATH=/opt/dcm2niix/bin:\$PATH; (note the semi-colon at the end)
- /opt/dcm2niix/bin/ (note the slash at the end)
- '"C:\\Program Files\\dcm2niix\\' (note the quotes to deal with the whitespace)
- args: Argument string that is passed to dcm2niix. Click [Test] and see the terminal output for usage

Tip: SPM users may want to use '-z n', which produces unzipped nifti's

2.5.3 Plugins

BIDScoin provides the possibility for researchers to write custom python functions that will be executed at bidsmapper and bidscoiner runtime. To use this functionality, enter the name of the module (default location is the plugins-folder; otherwise the full path must be provided) in the bidsmapper dictionary file to import the plugin functions. See [advanced usage](#) for more details.

2.6 Advanced usage

2.6.1 Site specific / customized template

If you want to convert many studies with similar acquisition protocols then you **may** consider (NB: this is in no way necessary) creating your own customized bidsmapper template. This template can then be passed to the [bidsmapper](#) tool (instead of the default [path_to_bidscoin]/heuristics/bidsmapper_template.yaml template) to automatically identify the different scans in your (future) studies and map these to the correct BIDS modalities.

Generally speaking, a bidsmap file contains a collection of key-value dictionaries that define unique mappings between different types (runs) of source data onto BIDS outcome data. As illustrated in the figure below, each run item in the bidsmap has a `provenance` key-value pair to store the pathname of a representative data sample of that run. Each run item also contains a source data `attributes` object, i.e. a key-value dictionary with keys and values that are extracted from the provenance data sample, as well as a `bids` object, i.e. a key-value dictionary that determines the filename of the BIDS output file. The different key-value pairs in the `attributes` dictionary represent properties of the source data and should uniquely identify the different runs in a session. But these attribute-values should not vary between sessions, making the length of the bidsmap only dependent on the acquisition protocol and not on the number of subjects and sessions in the data collection. The difference between a bidsmap template and the study bidsmap that comes out of the `bidsmapper` is that the template contains / defines the key-values that will be used by the `bidsmapper` and that the template contains all possible runs. The study bidsmap contains only runs that were encountered in the study, with key-values that are specific for that study. A bidsmap has different sections for different source data modalities, i.e. DICOM, PAR, P7, Nifti, FileSystem, as well as a section for the BIDScoin Options. Within each source data section there sub-sections for the different BIDS modalities, i.e. for `anat`, `func`, `dwi`, `fmap`, `pet`, `beh` and `extra_data`, and for the `participant_label` and `session_label`. It is important to note that BIDScoin tools, given a data sample, will go through the bidsmap (from top to bottom) until they come across a run with attribute values that match the attribute values of the data sample (NB: empty values are ignored). At that point a bidsmapping is made, i.e. the bids values will be taken to construct a BIDS output filename.

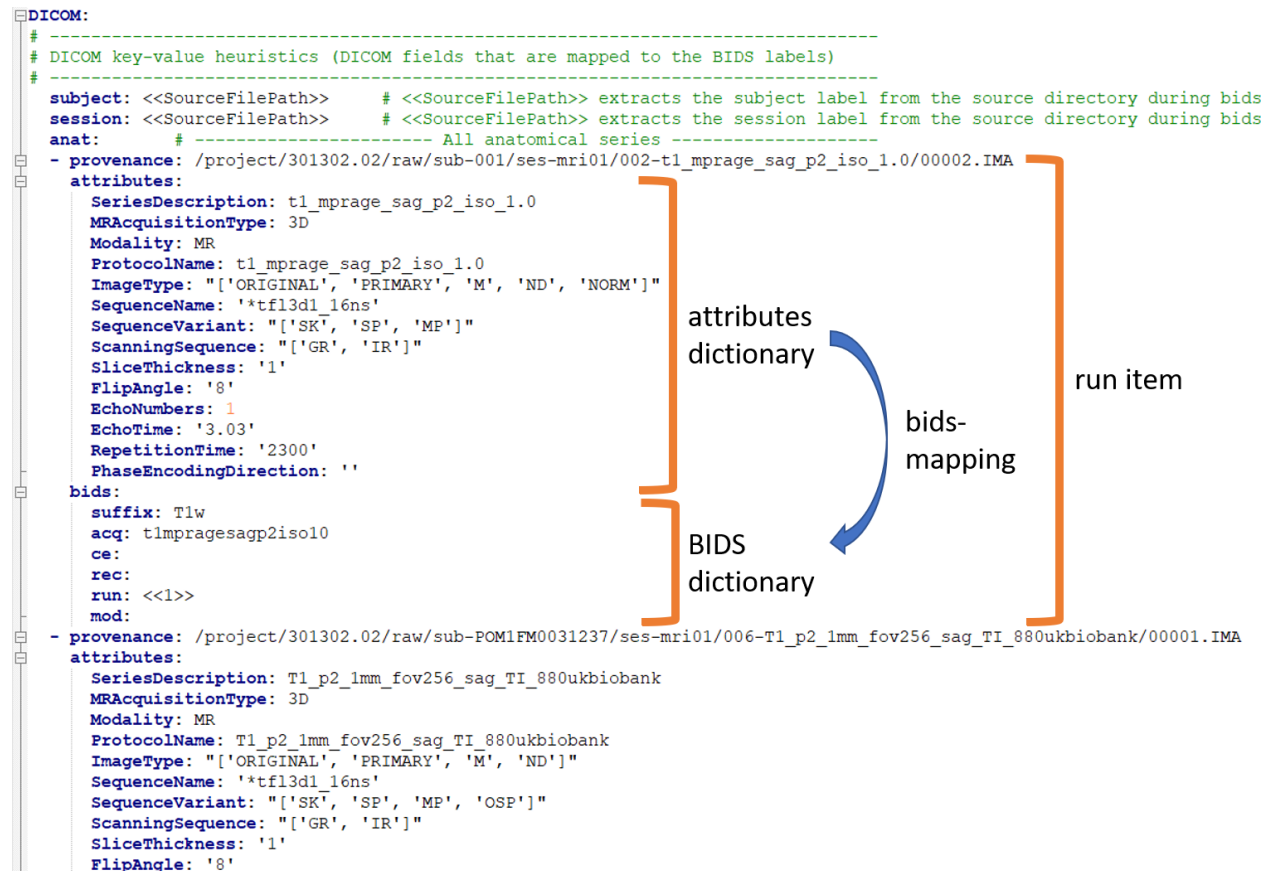


Fig. 2: A snippet of a study bidsmap, showing a DICOM section with a few run items in the `anat` subsection

To create your own template bidsmap you can best first make a copy of the default template (`[path_to_bidscoin]/heuristics/bidsmap_template.yaml`) or of the DCCN example template (`[path_to_bidscoin]/heuristics/bidsmap_dccn.yaml`) and customize that bidsmap to your needs:

1. **Using the `bidseditor`.** This is the easiest way to create a bidsmap template since it uses only a GUI and doesn't

require in-depth knowledge of bidsmaps and YAML files. If you have a run item in your study that you would like to be automatically mapped in other / future studies you can simply append that run to the standard or to a custom template bidsmap by editing it to your needs and click the **Export** button (see below). With the GUI you can still use advanced features, such as **Unix shell-style wildcards** in the values of the source attributes (see left panel), or such as using lists of attribute values (of which either one can match), or simply empty field to ignore the item. The main limitation of using the GUI is that the run items are always appended to a bidsmap template, meaning that they are last in line and will be used only if no other item in the template matches. It also means that like this you cannot edit the already existing run items in the bidsmap. Another (smaller) limitation is that with the GUI you cannot make usage of YAML anchors and references. Both limitations are overcome when directly editing the template bidsmap yourself using a text editor (see next point).

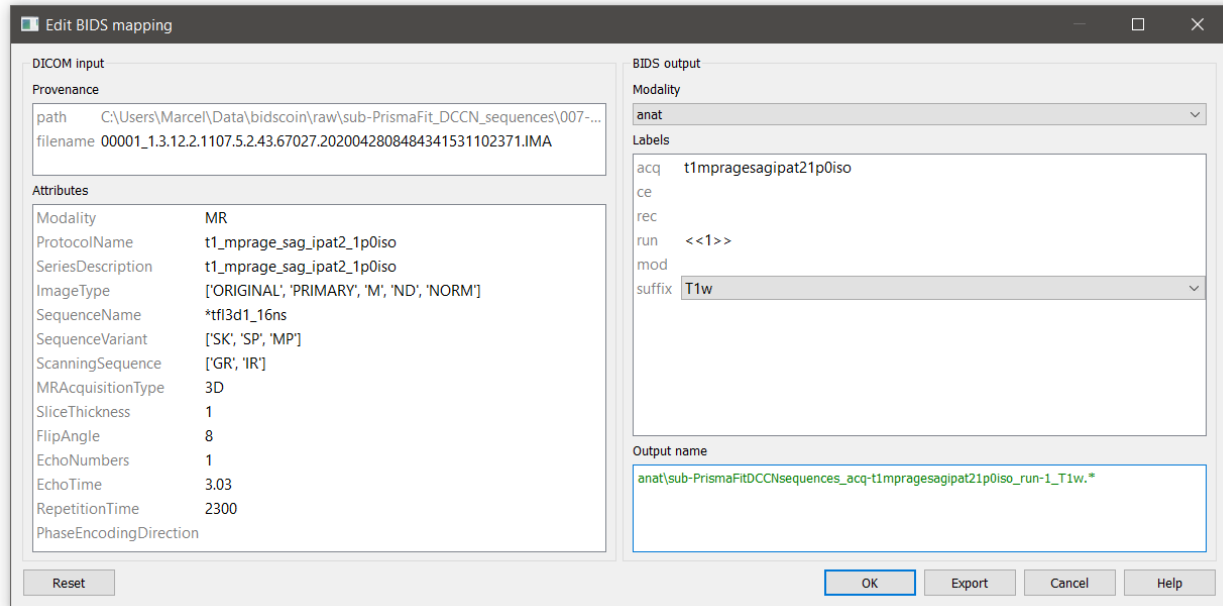


Fig. 3: The edit window with the option to export the customized mapping of run a item

2. **Using a text editor.** This is the most powerful way to create or modify a bidsmap template but requires more indepth knowledge of **YAML** and of how BIDScoin identifies different acquisitions in a protocol given a bidsmap. How you can customize your template is well illustrated by the DCCN template bidsmap ([path_to_bidscoin]/heuristics/bidsmap_dccn.yaml). If you open that template, there are a few things to take notice of (as shown in the template snippet below). First, you can see that the DCCN template makes use of **YAML anchors and aliases** (to make maintainance more sustainable). The second thing to notice is that, of the first run, all values of the attribute dictionary are empty, meaning that it won't match any run / will be ignored. In that way, however, the subsequent runs that alias (<<: *anatattributes_dicom) this anchor (&anatattributes_dicom) will inherit only the keys and can inject their own values, as shown in the second run. The first run of each modality sub-section (like anat) also serves as the default bidsmapping when users manually overrule / change the bids modality using the **bidsmapper** GUI. Finally, it is important to take notice of the usage of the lists of values (any of which can match) and **Unix shell-style wildcards** (see DICOM Attributes).

```

anat:          # ----- All anatomical runs -----
- provenance: ~          # The first run item with empty attributes will not
↪ match anything but will be used when changing modality in the bidseditor GUI ->
↪ suffix = T1w
  attributes: &anatattributes_dicom
  Modality: ~

```

(continues on next page)

(continued from previous page)

```

ProtocolName: ~
SeriesDescription: ~
ImageType: ~
SequenceName: ~
SequenceVariant: ~
ScanningSequence: ~
MRAcquisitionType: ~
SliceThickness: ~
FlipAngle: ~
EchoNumbers: ~
EchoTime: ~
RepetitionTime: ~
PhaseEncodingDirection: ~
bids: &anatbids_dicom
  acq: <SeriesDescription>      # A dynamic label which will be replaced during
↳bidscoiner runtime with the DICOM attribute value
  ce: ~
  rec: ~
  run: <<1>>                  # A dynamic label that will be increased during
↳bidscoiner runtime. NB: changing this value may lead to collisions / overwriting of
↳BIDS data
  mod: ~
  suffix: T1w
- provenance: ~              # The second run item with non-empty attributes (
↳'SeriesDescription' and 'MRAcquisitionType') will match any run with these
↳attribute values
  attributes:
    <<: *anatattributes_dicom
    SeriesDescription: ['*mprage*', '*MPRAGE*', '*MPRage*', '*t1w*', '*T1w*', '*T1W*']
    MRAcquisitionType: 3D
  bids:
    <<: *anatbids_dicom
    suffix: T1w

```

Snippet from the “`bidsmap_dcn.yaml`” template, showing a DICOM section with the first two run items in the anat subsection

2.6.2 Plugins

BIDScoin has the option to import plugins to further automate / complete the conversion from source data to BIDS. The plugin takes is called each time the BIDScoin tool has finished processing a run or session, with arguments containing information about the run or session, as shown in the plugin example code below. The functions in the plugin module should be named `bidsmapper_plugin` to be called by `bidsmapper` and `bidscoiner_plugin` to be called by `bidscoiner`.

```

import logging
from pathlib import Path

LOGGER = logging.getLogger(f'bidscoin.{Path(__file__).stem}')

def bidsmapper_plugin(seriesfolder: Path, bidsmap: dict, bidsmap_template: dict) -> dict:
    """
    The plugin to map info onto bids labels
    """

```

(continues on next page)

(continued from previous page)

```

:param seriesfolder:    The full-path name of the raw-data series folder
:param bidsmap:         The study bidsmap
:param bidsmap_template: Full BIDS heuristics data structure, with all options,
→ BIDS labels and attributes, etc
:return:               The study bidsmap with new entries in it
"""

LOGGER.debug(f'This is a bidsmapper demo-plugin working on: {seriesfolder}')
return bidsmap

def bidscoiner_plugin(session: Path, bidsmap: dict, bidsfolder: Path, personals: dict) → None:
    """
    The plugin to cast the series into the bids folder

    :param session:      The full-path name of the subject/session raw data source_
→ folder
    :param bidsmap:      The full mapping heuristics from the bidsmap YAML-file
    :param bidsfolder:   The full-path name of the BIDS root-folder
    :param personals:    The dictionary with the personal information
    :return:             Nothing
    """

    LOGGER.debug(f'This is a bidscoiner demo-plugin working on: {session} →
→ {bidsfolder}')

```

Plugin example code

2.7 Screenshots

2.7.1 The bidseditor

2.8 Demo and tutorial

2.8.1 BIDS introduction and BIDScoin demo

A good starting point to learn more about BIDS and BIDScoin is to watch [this presentation](#) from the OpenMR Benelux 2020 meeting ([slides](#)). The first 14 minutes [Robert Oostenveld](#) provides a general overview of the BIDS standard, after which [Marcel Zwiers](#) presents the design of BIDScoin and demonstrates hands-on how you can use it to convert a dataset to BIDS.

2.8.2 Hands-on tutorial

The following tutorial is somewhat tailored to the dataflow in the DCCN (e.g. using the module system to set the BIDScoin shell environment), but should nevertheless make the basic parts of the BIDScoin workflow clear for everyone.

1. **Preparation.** Activate the bidscoin environment and create a tutorial playground folder in your home directory by executing these bash commands (see also `module help bidscoin`):

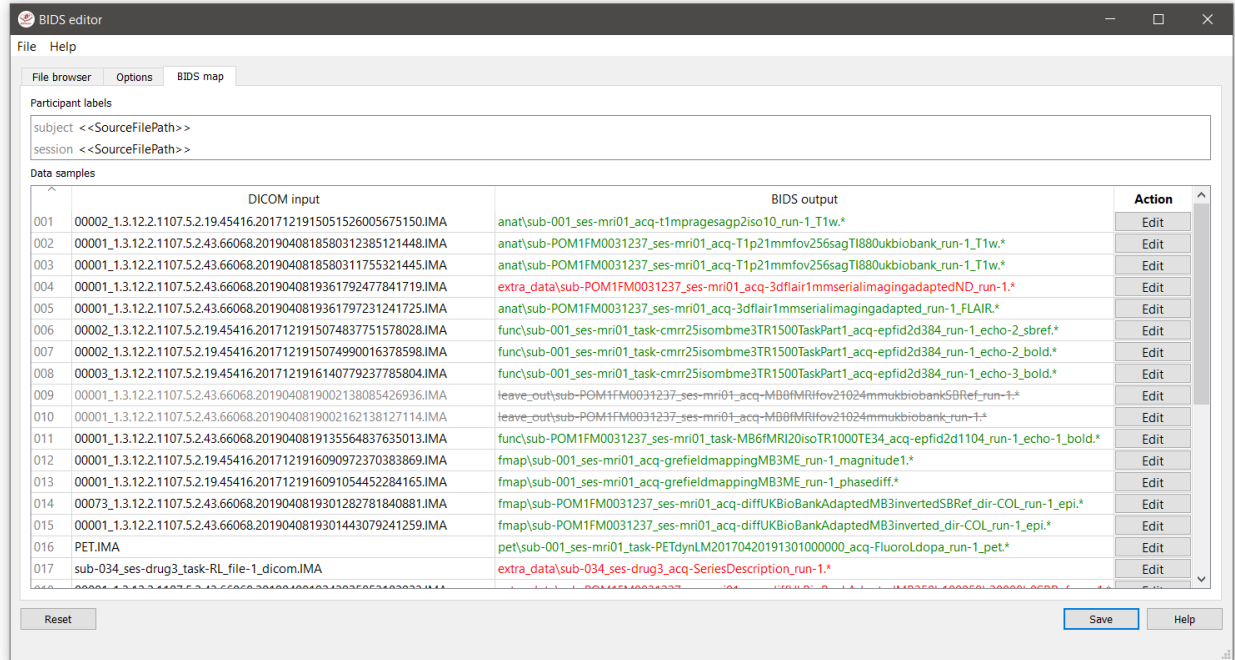


Fig. 4: The main window with the bidsmap overview

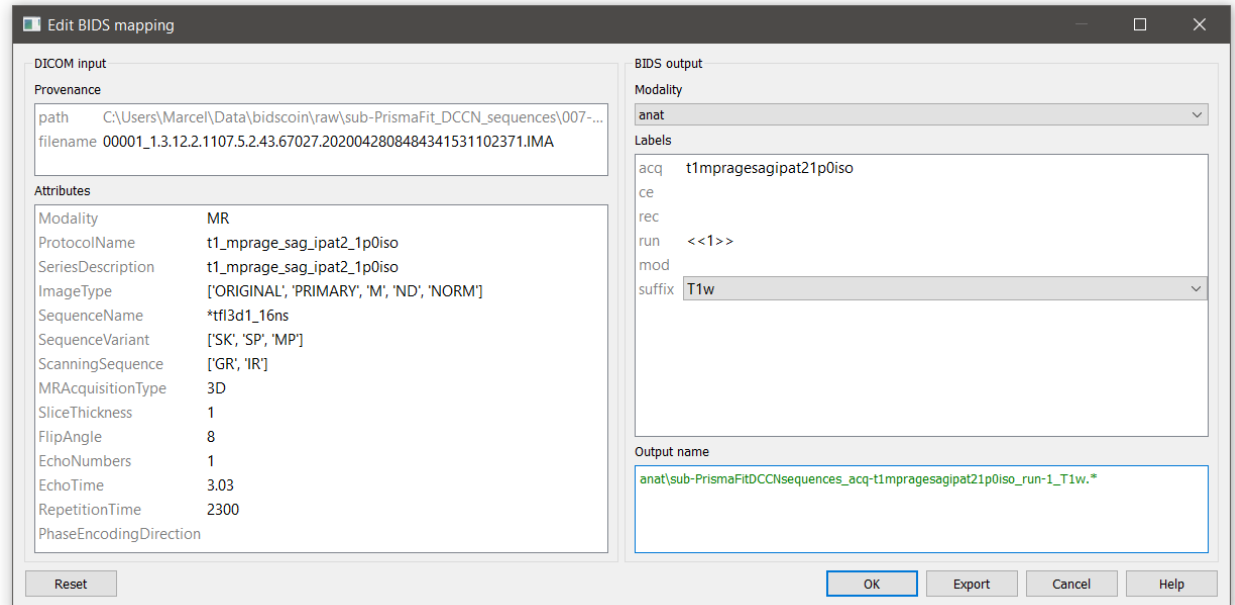


Fig. 5: The edit window for customizing the individual bidsmap entries

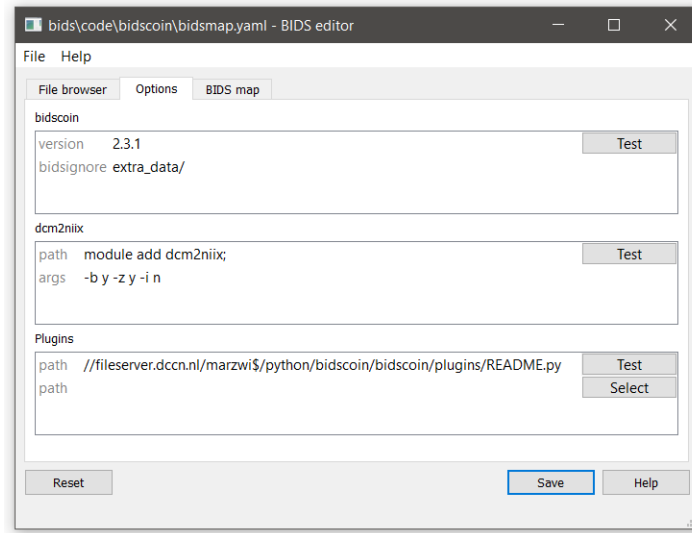


Fig. 6: The options window with BIDScoin settings

```
$ module add bidscoin
$ source activate /opt/bidscoin
$ pulltutorialdata
$ cd bidscointutorial
```

The new `bidscointutorial` folder contains a raw source-data folder and a `bids_ref` reference BIDS folder, i.e. the intended end product of this tutorial. In the raw folder you will find these DICOM series:

```
001-localizer_32ch-head          A localizer scan that is not scientifically_
↪relevant and can be left out of the BIDS dataset
002-AAHead_Scout_32ch-head      A localizer scan that is not scientifically_
↪relevant and can be left out of the BIDS dataset
007-t1_mprage_sag_ipat2_1p0iso   An anatomical T1-weighted scan
047-cmrr_2p4iso_mb8_TR0700_SBRef A single-band reference scan of the_
↪subsequent multi-band functional MRI scan
048-cmrr_2p4iso_mb8_TR0700      A multi-band functional MRI scan
049-field_map_2p4iso            The fieldmap magnitude images of the first_
↪and second echo. Set as "magnitude1", bidscoiner will recognize the format. This_
↪fieldmap is intended for the previous functional MRI scan
050-field_map_2p4iso            The fieldmap phase difference image of the_
↪first and second echo
059-cmrr_2p5iso_mb3me3_TR1500_SBRef A single-band reference scan of the_
↪subsequent multi-echo functional MRI scan
060-cmrr_2p5iso_mb3me3_TR1500   A multi-band multi-echo functional MRI scan
061-field_map_2p5iso            Idem, the fieldmap magnitude images of the_
↪first and second echo, intended for the previous functional MRI scan
062-field_map_2p5iso            Idem, the fieldmap phase difference image of_
↪the first and second echo
```

Let's begin with inspecting this new raw data collection:

- Are the DICOM files for all the `bids/sub-*` folders organised in series-subfolders (e.g. `sub-001/ses-01/003-T1MPRAGE/0001.dcm` etc)? Use `dicomsort` if this is not the case. A help text for all BIDScoin tools is available by running the tool with the `-h` flag (e.g. `rawmapper -h`)
- Use the `rawmapper` command to print out the DICOM values of the “EchoTime”, “Sex” and “AcquisitionDate”

of the fMRI series in the raw folder

2. **BIDS mapping.** Now we can make a `bidsmap`, i.e. the mapping from DICOM source-files to BIDS target-files. To that end, scan all folders in the raw data collection by running the `bidsmapper` command:

```
$ bidsmapper raw bids
```

- In the GUI that appears, rename the task and acquisition label of the functional scans into something more readable, e.g. `task-Reward` for the `acq-mb8` scans and “task-Stop” for the `acq-mb3me3` scans. Also make the name of the T1 scan more pleasant, e.g. by naming the acquisition label simply `acq-mprage`.
 - Add a search pattern to the `IntendedFor` field such that it will select your fMRI runs correctly (see the `bidseditor` `fieldmap` section for more details)
 - Since for this dataset we only have one session per subject, remove the session label (and note how the output names simplify, omitting the session subfolders and labels)
 - When all done, (re)open the `bidsmap.yaml` file and change the options such that you will get non-zipped nifti data (i.e. `*.nii` instead of `*.nii.gz`) in your BIDS data collection. You can use a text editor or, much better, run the `bidseditor` command line tool.
3. **BIDS coining.** The next step is very simple (and can be repeated when new data comes in) and just applies the previously made `bidsmap` when converting the source data into a BIDS collection. To do this run the `bidscoiner` commandline tool (note that the input is the same as for the `bidsmapper`):

```
$ bidscoiner raw bids
```

- Check your `bids/code/bidscoin/bidscoiner.log` (the complete terminal output) and `bids/code/bidscoin/bidscoiner.errors` (the summary that is also printed at the end) files for any errors or warnings. You shouldn’t have any :-)
 - Compare the results in your `bids/sub-*` subject folders with the in `bids_ref` reference result. Are the file and folder names the same (don’t worry about the multi-echo images and the `extra_data` images, they are combined/generated as described below)? Also check the json sidecar files of the fieldmaps. Do they have the right “EchoTime” and “IntendedFor” fields?
 - What happens if you re-run the `bidscoiner` command? Are the same subjects processed again? Re-run “sub-001”.
4. **Finishing up.** Now that you have converted the data to BIDS, you still need to do some manual work to make it fully ready for data analysis and sharing
 - Combine the echos using the `echocombine` tool, such that the individual echo images are replaced by the ech-combined image
 - Deface the anatomical scans using the `echocombine` tool. This will take a while, but will obviously not work well for our phantom dataset. Therefore store the ‘defaced’ output in the `extra_data` folder (instead of e.g. overwriting the existing images)
 - Inspect the `bids/participants.tsv` file and decide if it is ok.
 - Update the `dataset_description.json` and `README` files in your `bids` folder
 - As a final step, run the `bids-validator` on your `~/bids_tutorial` folder. Are you completely ready now to share this dataset?