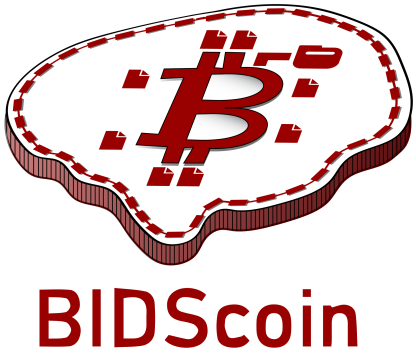
# BIDScoin

***Release 3.7.0-dev***

**May 31, 2021**

# Contents




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 then depending on complex or ambiguous programmatic logic for source data-type identification, BIDScoin uses a mapping approach to identify and convert the raw source data types into BIDS data. Different runs of source data are identified by way of file system properties (e.g. file size) and/or from file attributes (e.g. `ProtocolName` from the DICOM header). The mapping information about how these runs should be converted to BIDS can be pre-specified (e.g. per site), which should provide for a good first automatic data-typing and mapping, using all information available on disk. Then the researcher can interactively check and edit this mapping – bringing in the missing knowledge that often exists only in his or her head!

Because all the mapping information can be easily edited with the 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

- [x] DICOM source data
- [x] PAR / REC source data (Philips)
- [x] Physiological logging data*
- [x] Fieldmaps*
- [x] Multi-echo data*
- [x] Multi-coil data*
- [x] PET data*
- [ ] Stimulus / behavioural logfiles
- [x] Plug-ins
- [x] Defacing
- [x] Multi-echo combination

```
* = Only DICOM source data / tested for Siemens
```

```
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.
```

CHAPTER 2

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 well tested) that satisfy the system requirements:

- dcm2niix
- python 3.8 or higher
- (FSL, optional and only needed when using the defacing tool to remove facial features from anatomical scans)

### 2.1.1 Dcm2niix installation

The default `dcm2niix2bids` plugin relies on dcm2niix to convert DICOM and PAR/REC files to nifti. To use this plugin, 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 Options section in your bidsmap or, for once and for all, put it in your Site specific / customized template bidsmap.

### 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 and OS-X 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.

### 2.1.3 BIDScoin installation

To install BIDScoin on your system run the following command in a command-terminal (institute users may want to create and 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 --upgrade git+https://github.com/Donders-Institute/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:

```
$ pip install ./bidscoin
```

#### Testing BIDScoin

You can run the central `bidscoin` tool to test your installation:

```
$ bidscoin -t
```

#### Updating BIDScoin

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

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

**Caution:**

- The bidsmaps are not garanteed 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]/data` structure (the `ses-identifier` subfolder is optional). The data folder 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
|   |   |   |-- 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 the DICOM files are stored. DICOMDIRs are often used in clinical settings and may look like:

```
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 organization is sometimes used when exporting data in clinical settings (the session sub-folder is optional):

```
sourcedata
|-- sub-001
|   `-- ses-mri01
|       |-- IM_0001.dcm
|       |-- IM_0002.dcm
|       |-- 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 in one directory. This organization is how users often export their data from Philips scanners in research settings (the session sub-folder is optional):

```
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 your session data in any of the above data organizations as zipped (`.zip`) or tarzipped (e.g. `.tar.gz`) archive files. BIDScoin workflow tools will automatically unpack/unzip those archive files in a temporary folder and then process your session‘‘ data‘‘ from there. For flat/DICOMDIR data, BIDScoin tools will automatically run *dicomsort* in a temporary folder to sort them in seriesfolders. BIDScoin tools that work from a temporary folder has the downsde of getting a speed penalty.

---

**Tip:** BIDScoin will skip (linux-style hidden) files and folders starting with a . (dot) character. You can use this feature to flexibly omit subjects, sessions or runs from your bids repository, for instance when you restarted a MRI scan because something went wrong with the stimulus presentation and you don’t want that data to be converted and

enumerated as *run-1*, *run-2*.

### 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. 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 icomsort 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
```

(continues on next page)

(continued from previous page)

```
                        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
```

(continues on next page)

(continued from previous page)

```
                        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

With a sufficiently organized source data folder, the data conversion to BIDS can be performed by running the *(1a)* the bidsmapper, *(1b)* the bidseditor and *(2)* the bidscoiner command-line tools. The bidsmapper starts by building a map of the different kind of data types (scans) in your source dataset, which you can then edit with the bidseditor. The bidscoiner reads this so-called study bidsmap, which tells it how exactly to convert ("coin") the source data into a BIDS data repository.

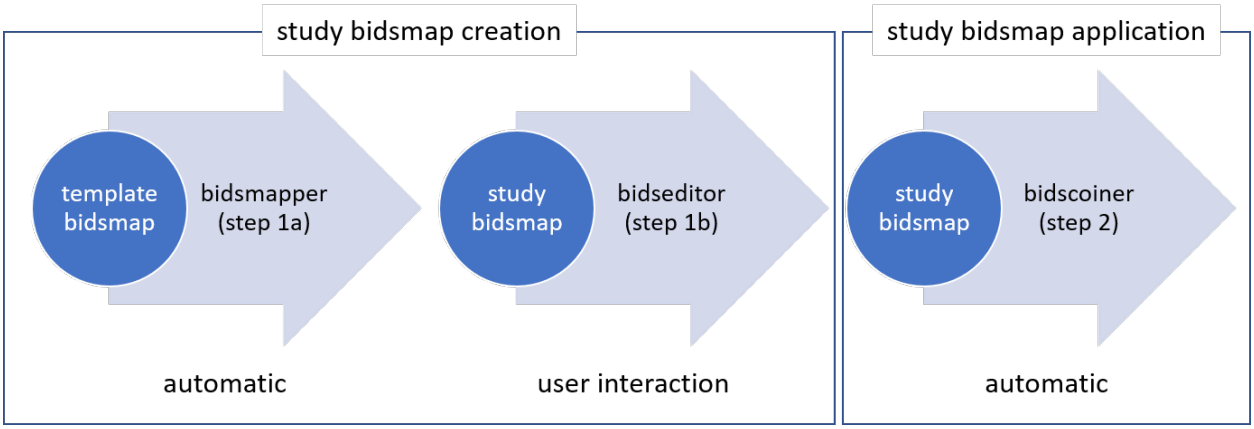



Fig. 1: Creation and application of a study bidsmap

By default, 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
```

If you add new subjects all you need to do is re-run the bidscoiner – unless the scan protocol was changed, then you also need to first re-run the bidsmapper to add the new samples to the study bidsmap. The paragraphs below describe

the BIDScoin worklow in more detail.

### 2.3.1 Step 1a: Running the bidsmapper

```
usage: bidsmapper [-h] [-b BIDSMAP] [-t TEMPLATE] [-n SUBPREFIX] [-m SESPREFIX] [-s]␣
↪[-a] [-f] [-v]
                  sourcefolder bidsfolder

The bidsmapper scans your source data repository to identify different data types by␣
↪matching
them against the items in the template bidsmap. Once a match is found, a mapping to␣
↪BIDS output
data types is made. You can check and edit these generated bids-mappings to your␣
↪needs with the
(automatically launched) bidseditor. Re-run the bidsmapper whenever something was␣
↪changed in
your data acquisition protocol and edit the new data type to your needs (your␣
↪existing bidsmap
will be re-used).

The bidsmapper uses plugins, as stored in the bidsmap['Options'], to do the actual␣
↪work

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

optional arguments:
  -h, --help            show this help message and exit
  -b BIDSMAP, --bidsmap BIDSMAP
                        The study bidsmap file with the mapping 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 file 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_dccn.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 provenance data samples in the
                        bidsfolder/'code'/'provenance' folder (useful for inspecting␣
↪e.g. zipped
                        or transfered datasets)
```

(continues on next page)

(continued from previous page)

```
  -a, --automated       Flag to save the automatically generated bidsmap to disk and␣
↪without
                        interactively tweaking it with the bidseditor
  -f, --force           Flag to discard the previously saved bidsmap and logfile
  -v, --version         Show the installed version and check for updates

examples:
  bidsmapper /project/foo/raw /project/foo/bids
  bidsmapper /project/foo/raw /project/foo/bids -t bidsmap_template
```

After the source data has been scanned, the bidsmapper will automatically launch *step 1b*. For a fully automated workflow users can skip this interactive step using the `-i` option (see above).

**Tip:** The default template bidsmap (`-t bidsmap_dccn`) is customized for acquisitions at the DCCN. If this bidsmap is not working well for you, consider adapting it to your needs so that the bidsmapper can recognize more of your scans and map them to BIDS the way you prefer.

### 2.3.2 Step 1b: Running the bidseditor

```
usage: bidseditor [-h] [-b BIDSMAP] [-t TEMPLATE] [-n SUBPREFIX] [-m SESPREFIX]␣
↪bidsfolder

This tool launches a graphical user interface for editing the bidsmap that is␣
↪produced by the
bidsmapper. You can edit the BIDS data types and entities until all data-samples have␣
↪a meaningful
and nicely readable BIDS output name. The (saved) bidsmap.yaml output file will be␣
↪used by the
bidscoiner to do the conversion conversion of the source data to BIDS.

You can hoover with your mouse over items to get help text (pop-up tooltips).

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
                        The study bidsmap file with the mapping 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 template bidsmap file 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_dccn.yaml
  -n SUBPREFIX, --subprefix SUBPREFIX
                        The prefix common for all the source subject-folders.␣
↪Default: 'sub-'
```

(continues on next page)

(continued from previous page)

```
  -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_template.yaml
  bidseditor /project/foo/bids -b my/custom/bidsmap.yaml
```

## Main window

As shown below, the main window of the bidseditor opens with `DICOM mappings` and `PAR mappings` tabs, each containing of a `Participant labels` table and a *Data samples* table. By default, the participant table contains dynamic `<<SourceFilePath>>` labels, which makes that the label is extracted / copied from the path of the source data during bidscoiner runtime. Alternatively, you can put a dynamic attribute label here (e.g. <<PatientName>>) if you want to extract that information from the source header. The data samples table shows a list of input files (left side) that uniquely represent all the different data types in the sourcedata repository, in conjunction with a preview of their `BIDS output` names (right side). The BIDS output names are shown in red if they are not BIDS compliant, striked-out gray when the runs will be ignored / skipped in the conversion to BIDS, otherwise it is colored green.

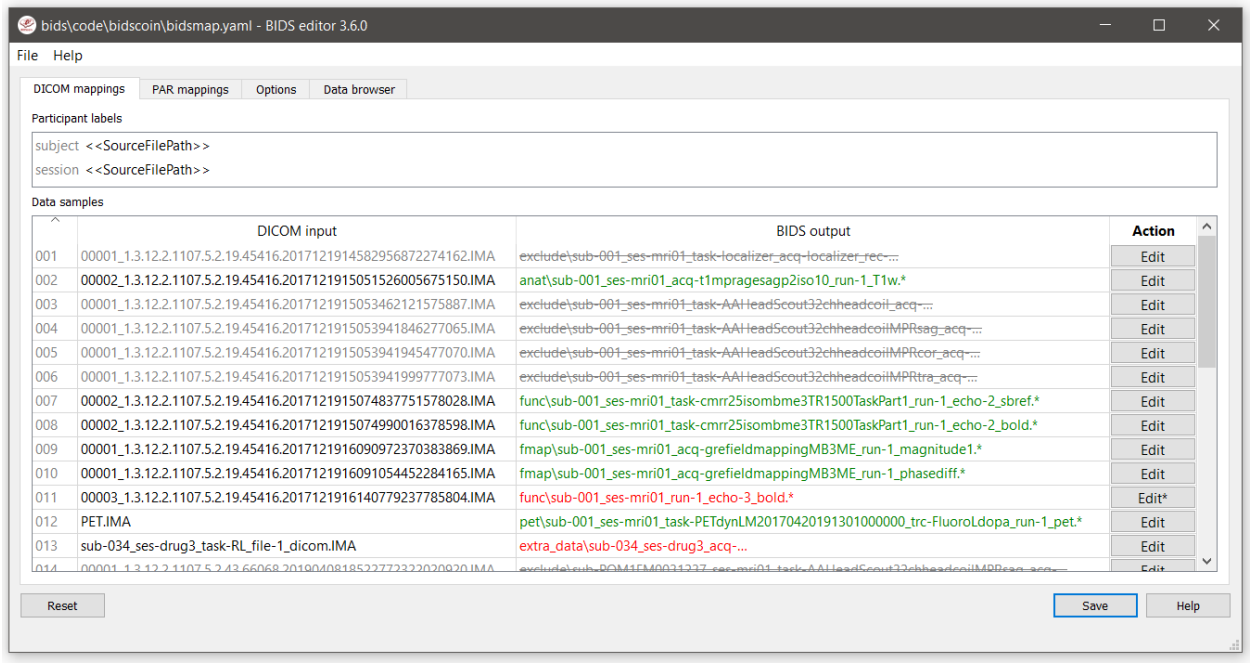



Fig. 2: The main window with an overview of all the bidsmap run items

**Tip:** Clear the `session` label field if you have data with only one seesion. This will remove the optional session label from the BIDS ouptput name

### Edit window

In the main window, you can double-click the BIDS output name of a data sample or click the `Edit` button next to it (NB: the * in this button indicates that attention is required) to open a second window, as shown below. In this edit window, the full bids-mapping info of the clicked data-sample (AKA run-item) is shown, with the filesystem `Properties` and file `Attributes` input on the left, and, most importantly, the associated BIDS `Data type`, `Data filename` and `Meta data` output on the right. By double-clicking cells, you can modify the (automatically generated) values that you think are not optimal or incorrect. You should first make sure the BIDS `Data type` (drop down menu) and its `suffix` label (drop down menu) are set correctly, after which the BIDS entities can be edited. Each time an item is edited, a new `Data filename` preview is shown (green or red text indicates that the name is BIDS compliant or not). In the `Meta data` table (see the figure below) you can enter key-value pairs that you like to to be appended (by the standard `dcm2niix2bids` plugin) to the standard meta-data in the json sidecar file. Editing the source properties and attributes of a study bidsmap is usually not necessary and considered advanced usage.

If the preview of the BIDS filename and meta-data both look good, you can store the data in the bidsmap by clicking the `OK` button.

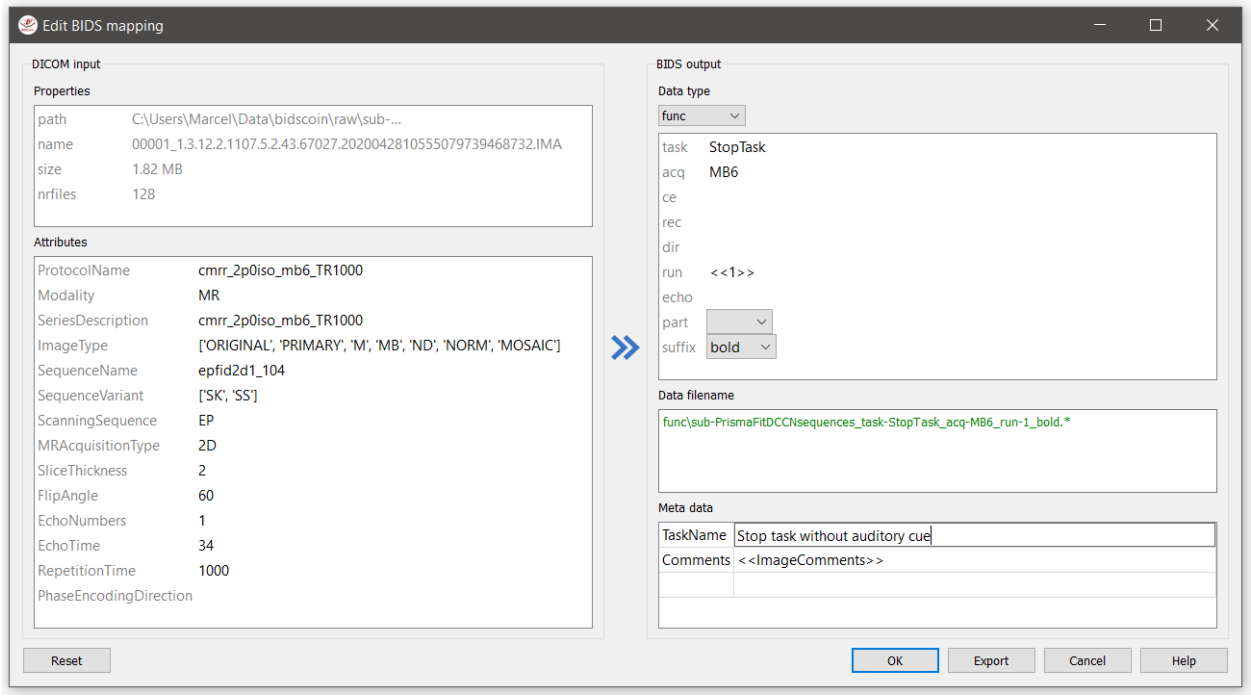



Fig. 3: The edit window for customizing a bidsmap run item, featuring the `TaskName` value being set to something more informative

Finally, if all BIDS output names in the main window are fine, you can click on the `Save` button and proceed with running the bidscoiner tool. Note that the bidsmapper and bidseditor don't do anything except reading from and writing to a bidsmap yaml-file.

---

**Tip:** The BIDScoin GUI features several ways to help you setting the right values: * Double-clicking an input filename pops-up an inspection window with the full header information (e.g. useful for checking attributes that are not (yet) in your bidsmap) * Hoovering with your mouse over a cell pops-up a tooltip with more background information (e.g. from the BIDS specifications) * Always check the terminal output and make sure there are no warnings or error messages there (a summary of them is printed when exiting the application)

---

**Note:** **Fieldmaps** are acquired and stored in various (sequences and manufacturer dependent) ways and may require special treatment. For instance, it could be that you have `magnitude1` and `magnitude2` data in one series-folder (which is what Siemens can do). In that case you should select the `magnitude1` suffix and let bidscoiner automatically pick up the other magnitude image during runtime. The same holds for `phase1` and `phase2` data. The suffix `magnitude` can be selected for sequences that save fielmaps directly. See the BIDS specification for more details on fieldmap suffixes.

Fieldmaps are typically acquired to be applied to specific other scans from the same session. If this is the case then you should indicate this in the `IntendedFor` meta-data field, either using a single search string or multiple dynamic strings to select the runs that have that string pattern in their BIDS file name. For instance you can use `task` to select all functional runs or use `<<Stop*Go><Reward>>` to select "Stop1Go"-, "Stop2Go"- and "Reward"-runs. NB: bidsapps may not use your fieldmap at all if you leave this field empty!

### 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") your source datasets to nifti / json / tsv BIDS datasets using
the information from the bidsmap.yaml file. Edit this bidsmap to your needs using the
bidseditor tool before running this function or (re-)run the bidsmapper whenever you
encounter unexpected data. You can run bidscoiner after all data has been collected,
or run / re-run it whenever new data has been added to your source folder (presuming
the scan protocol hasn't changed). Also, if you delete a subject/session folder from
the bidsfolder, it will simply be re-created from the sourcefolder the next time you
run the bidscoiner.

The bidscoiner uses plugins, as stored in the bidsmap['Options'], to do the actual␣
↪work

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

positional arguments:
  sourcefolder          The study root folder containing the raw data in
                        sub-#/[ses-#/]data subfolders (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␣
↪[PART
ICIPANT_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
```

(continues on next page)

(continued from previous page)

```
                        will not be processed (also when the --force flag is given).
                        Otherwise the participants.tsv table is ignored
  -b BIDSMAP, --bidsmap BIDSMAP
                        The study bidsmap file with the mapping 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 installed version and check for updates

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

**Tip:**

- Always check the terminal output for possible warnings or errors (a summary of them is printed at the end)
- 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 The bidsmap explained

### 2.4.1 Structure and content

A central concept in BIDScoin is the so-called bidsmap. Generally speaking, a bidsmap is a collection of run-items that define how source data (e.g. a T1w- or a T2w-scan) should be converted to BIDS output data. As illustrated in the figure below (but see also the screenshot of the edit window), a run-item consists of a `provenance` field and a `filesystem`, `attributes`, `bids` and a `meta` dictionary (a dictionary is a set of key-value pairs):

1. The provenance field contains the pathname of a source data sample that is representative for the run-item. The provenance data is not strictly necessary but very useful for deeper inspection of the source data and for tracing back the process, e.g. in case of encountering unexpected results
2. The filesystem dictionary contains file system properties of the data sample, such as its filename or the disk space that it occupies. Depending on your data management, this information allows or can help to identify different datatypes in your source data repository
3. The attributes dictionary contains attributes from the source data itself, such as the 'ProtocolName' from the DICOM header. The source attributes are a very rich source of information that is normally sufficient to identify the different datatypes in your source data repository

4. The bids dictionary contains the BIDS datatype and entities that determine the filename of the BIDS output data. The values in this dictionary can be freely edited by the user
5. The meta dictionary contains custom key-value pairs that are added to the json sidecar file by the bidscoiner (dcm2niix2bids). And because such data often varies from session to session, this dictionary typically contains dynamic attribute values that are evaluated during bidscoiner runtime (see the *special features* below)

So a run-item contains a single bids-mapping that links (2) and (3) to (4) and (5).

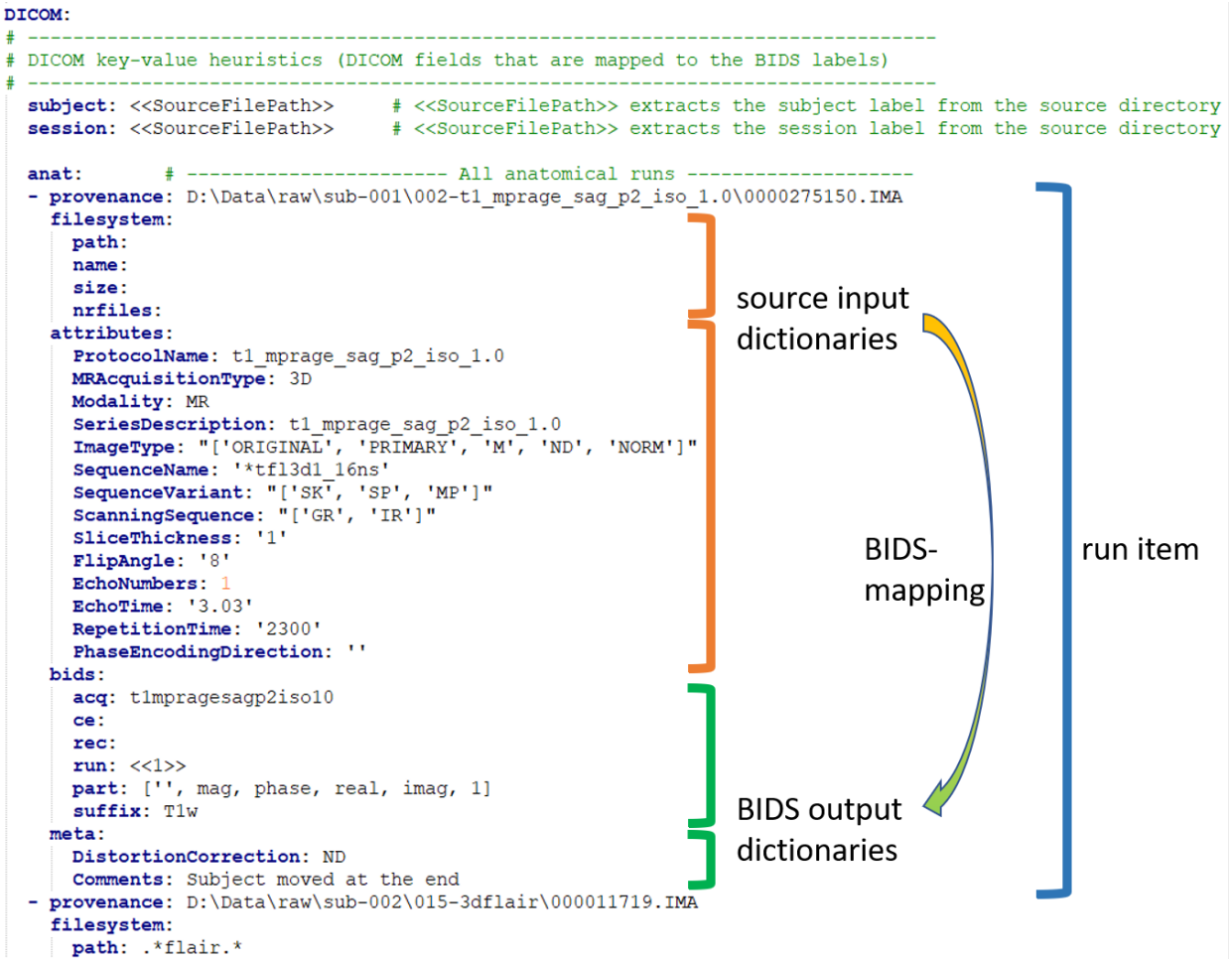



Fig. 4: A raw snippet of a study bidsmap file, showing a `DICOM` section with a few run-items in the `anat` subsection

A bidsmap is hierarchically organised in `DICOM` and `PAR` source modality sections, which in turn contain subsections for the‘‘participant_label‘‘ and `session_label`, for the BIDS datatypes (`fmap`, `anat`, `func`, `perf`, `dwi`, `pet`, `meg`, `eeg`, `ieeg`, `beh`) and for the `extra_data` and `exclude` datatypes. The particpicant- and session-label subsections are common to all run-items and contain key-value pairs that identify the subject and session labels. The datatype subsections list all the different run-items in the bidsmap. Next to the two source modality sections there is a general `Options` section, that accommodate customized BIDScoin and plugin settings and tweaks.

When BIDScoin tools process source data, they will take a sample of the data and the bidsmap, and scan through the datatype lists of run-items until they come across a run-item with filesystem and attribute values that match with the data sample at hand. At that point a bidsmapping is established and the bidsname and meta data for that sample can be generated. Within a datatype, run-items are matched from top to bottom, and scan order between datatypes is ‘exclude’, ‘fmap’, ‘anat’, ‘func’, ‘perf’, ‘dwi’, ‘pet’, ‘meg’, ‘eeg’, ‘ieeg’, ‘beh’ and ‘extra_data’. The ‘exclude’ datatype contains run-items for source data that need to be omitted when converting the source data to BIDS and

the 'extra_data' datatype contains run-items for including miscellaneous data that is not (yet) defined in the BIDS specifications. Bidsmaps can contain an unlimited number of run-items, including multiple run-items mapping onto the same BIDS target (e.g. when you renamed your DICOM scan protocol halfway your study and you don't want that irrelevant change to be reflected in the BIDS output).

### 2.4.2 From template to study

In BIDScoin a bidsmap can either be a template bidsmap or a study bidsmap. The difference between the two is that a template bidsmap is a comprehensive set of pre-defined run-items and serves as an input for the bidsmapper (see below) to automatically generate a first instantiation of a study bidsmap, containing just the matched run-items. Empty attribute values of the matched run-item will be expanded with values from the data sample, making the run-item much more specific and sensitive to small changes in the scan protocol. Users normally don't have to know about or interact with the template bidsmap, but they can create their own customized template. The study bidsmap can be interactively edited by the bidseditor before feeding it to the bidscoiner, but it is also possible (but not recommended) to skip the editing step and convert the data without any user interaction.

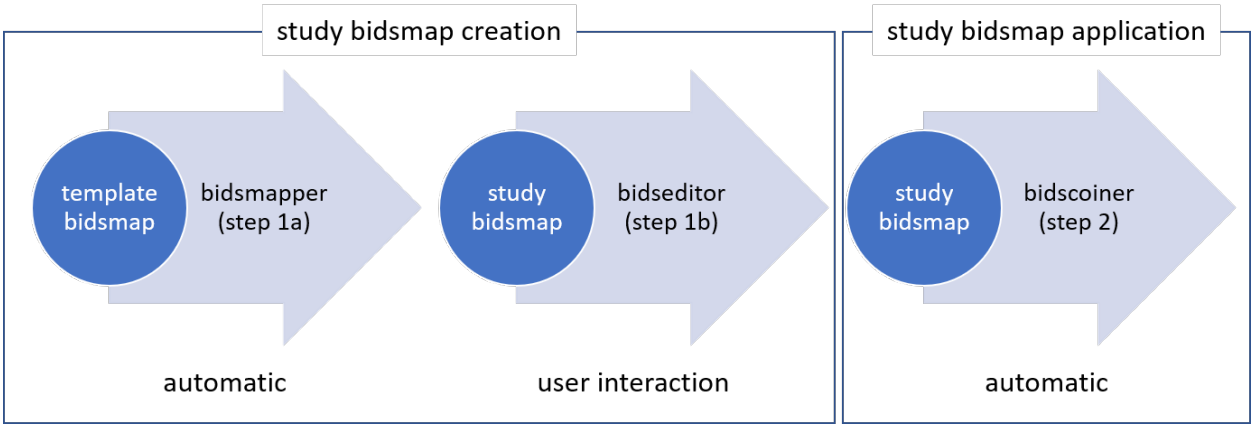



Fig. 5: Creation and application of a study bidsmap

### 2.4.3 Special bidsmap features

The dictionary values in a bidsmap are not simple strings but have some special features that make BIDScoin powerful, flexible and helpful:

- **Source matching patterns**. Source property and attribute values are regular expression patterns to run matching. For instance you can use `SeriesDescription: '.*MPRAGE.*'` to match all MPRAGE DICOM series as they come from your MRI scanner. This feature is useful for template bidsmaps.
- **Dynamic values**. Dictionary values can be static, in which case the value is just a normal string, or dynamic, when the string is enclosed with pointy brackets. In case of single pointy brackets the bids value will be replaced / expanded during bidsmapper, bidseditor and bidscoiner runtime by the value of the source attribute. For instance `acq: <MRAcquisitionType><SeriesDescription>` will be replaced by `acq: 3DMPRAGE`. In case of double enclosed pointy brackets, the value will be replaced only during bidscoiner runtime – this is useful for bids values that are subject/session dependent. For instance `run: <<1>>` will be replaced with `run: 1` or e.g. increased to `run: 2` if a file for that subject with that bidsname already exists. Dynamic values are also useful for meta data that is subject or session specific, such <<ImageComments>> or <<RadionuclideTotalDose>>, but not saved by default in the json sidecar files.
- **Bids value lists**. Instead of a normal string, a bids dictionary value can also be a list of strings, with the last list item being the (zero-based) list index that selects the actual value from the list. For instance the list `part:`

`['', 'mag', 'phase', 'real', 'imag', 2]` would select 'phase' as the value belonging to 'part'. A bids value list is made visible in the bidseditor as a drop-down menu in which the user can select the value (i.e. set the list index).

## 2.5 Finishing up

After a successful run of bidscoiner, the work to convert your data in a fully compliant BIDS dataset is usually not fully over and, depending on the complexity of your data-set, additional tools may need to be run to post-process (e.g. deface) your data or convert datatypes not supported by the standard BIDScoin plugins (e.g. EEG data). Below you can find some tips and additional BIDScoin tools that may help you finishing up.

### 2.5.1 Adding more meta-data

To make your dataset reproducable and shareable, you should add study-level meta-data in the modality agnostic BIDS files (BIDScoin saves stub versions of them). 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 `*_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 option and write their own log-file parser).

### 2.5.2 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: mecombine [-h] [-o OUTPUTNAME] [-a {PAID,TE,average}] [-w [WEIGHTS [WEIGHTS ...
↪]]] [-s] [-v VOLUMES]
                 pattern

Combine multi-echo echoes.

Tools to combine multiple echoes from an fMRI acquisition.
It expects input files saved as NIfTIs, preferably organised
according to the BIDS standard.

Currently three different combination algorithms are supported, implementing
the following weighting schemes:

1. PAID => TE * SNR
2. TE => TE
3. Simple Average => 1

positional arguments:
  pattern               Globlike search pattern with path to select the echo images␣
↪that need to be combined.
                        Because of the search, be sure to check that not too many␣
↪files are being read

optional arguments:
```

(continues on next page)

(continued from previous page)

```
  -h, --help            show this help message and exit
  -o OUTPUTNAME, --outputname OUTPUTNAME
                        File output name. If not a fullpath name, then the output␣
↪will be stored in the same
                        folder as the input. If empty, the output filename will be␣
↪the filename of the first
                        echo appended with a '_combined' suffix (default: )
  -a {PAID,TE,average}, --algorithm {PAID,TE,average}
                        Combination algorithm. Default: TE (default: TE)
  -w [WEIGHTS [WEIGHTS ...]], --weights [WEIGHTS [WEIGHTS ...]]
                        Weights (e.g. = echo times) for all echoes (default: None)
  -s, --saveweights     If passed and algorithm is PAID, save weights (default: False)
  -v VOLUMES, --volumes VOLUMES
                        Number of volumes that is used to compute the weights if␣
↪algorithm is PAID (default:
                        100)

examples:
  mecombine '/project/number/bids/sub-001/func/*_task-motor_*echo-*.nii.gz'
  mecombine '/project/number/bids/sub-001/func/*_task-rest_*echo-*.nii.gz' -a PAID
  mecombine '/project/number/bids/sub-001/func/*_acq-MBME_*run-01*.nii.gz' -w 11 22␣
↪33 -o sub-001_task-stroop_acq-mecombined_run-01_bold.nii.gz
```

### Defacing

Before sharing or pre-processing their images, users may want to deface their anatomical MRI acquisitions to protect the privacy of their subjects. The `deface`-tool is a wrapper around pydeface that writes BIDS valid output. NB: pydeface requires FSL to be installed on the system.

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

A wrapper around the 'pydeface' defacing tool (https://github.com/poldracklab/
↪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 specified then all sub-folders in the bidsfolder␣
↪will be processed
                        (default: None)
```

(continues on next page)

(continued from previous page)

```
  -o {fmap,anat,func,perf,dwi,meg,eeg,ieeg,beh,pet,extra_data,derivatives}, --output
↪{fmap,anat,func,perf,dwi,meg,eeg,ieeg,beh,pet,extra_data,derivatives}
                        A string that determines where the defaced images are saved.␣
↪It can be the name of a
                        BIDS datatype 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=2gb)
  -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* -c -n "-l walltime=00:60:00,mem=4gb"
  deface /project/3017065.01/bids anat/*_T1w* -a '{"cost": "corratio", "verbose": ""}'
```

### 2.5.3 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).

---

**Note:** Privacy-sensitive source data samples may be stored in `[bidsfolder]/code/bidscoin/provenance` (see the `-s` option in the bidsmapper).

---

## 2.6 Options

BIDScoin has different options and settings (see below) that can be adjusted per study bidsmap or, when you want to customize the default, edited in the template bidsmap. There are seperate settings for BIDScoin and for the individual plugins that can be edited by double clicking. Installed plugins can be removed or added to extend BIDScoin's functionality.

### 2.6.1 BIDScoin

These setting can be used by all the BIDScoin tools:

- `version`: Used to check for version conflicts (should correspond with the version in ../bidscoin/version.txt)
- `bidsignore`: Semicolon-separated list of datatypes that you want to include but that do not pass a BIDS validation test. Example: `bidsignore:  extra_data/;rTMS/;myfile.txt;yourfile.csv`

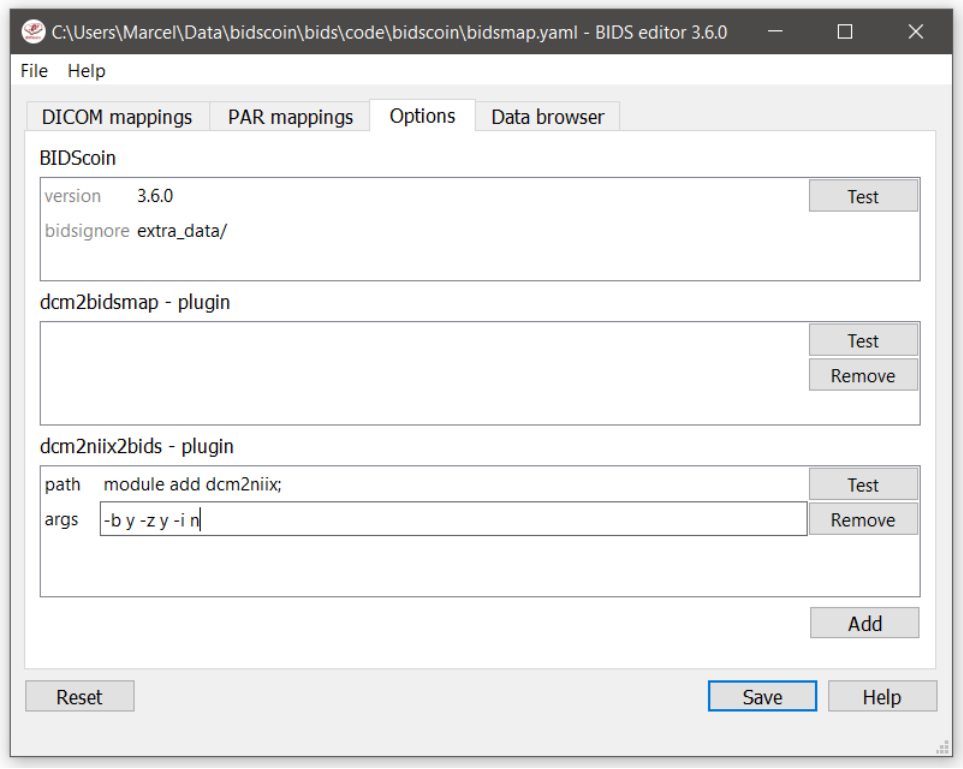



Fig. 6: The bidseditor options window with the different BIDScoin settings

### 2.6.2 dcm2bidsmap - plugin

The default bidsmapper plugin that builds a bidsmap from DICOM and PAR/REC source data. There are no settings for this plugin

### 2.6.3 dcm2niix2bids - plugin

The default bidscoiner plugin that converts DICOM and PAR/REC source data to BIDS-valid nifti- and json sidecar files. This plugin relies on dcm2niix, for which you can set the following options:

- `path`: A string that is prepended to the dcm2niix command to make sure it can be found by the operating system. You can leave it empty if dcm2niix is already on your shell path and callable from the command-line, otherwise you could use e.g.:
  - `module add dcm2niix/v1.0.20210317;` (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, e.g. `-b y -z n -i n`. Click [Test] and see the terminal output for usage

**Tip:**

- Put your custom dcm2niix path-setting in your template so that you don't have to set it anymore for every new study
- SPM users may want to use '-z n', which produces unzipped nifti's

## 2.7 Advanced usage

### 2.7.1 Site specific / customized template

The run-items in the default 'bidsmap_dccn' template bidsmap have source dictionary values that are tailored to MRI acquisitions in the Donders Institute. Hence, if you are using different protocol parameters that do not match with the template values, then your runs will initially be data (mis)typed by the bidsmapper as miscellaneous `extra_data` – which you then need to correct afterwards yourself. To improve that initial data typing and further automate your workflow, you may consider creating and using your own customized template bidsmap.

**Tip:** Make a copy of the DCCN template (`[path_to_bidscoin]/heuristics/bidsmap_dccn.yaml`) as a starting point for your own template bidsmap, and adapt it to your environment

**Note:** If you want to use different source attributes than the default set, then beware that the attribute values should not vary between different repeats of the data acquision. Otherwise the number of run-items in the bidsmap will not be a unique shortlist of the acquisition protocols in your study, but will instead become a lengthy list that is proportional to the number of subjects and sessions.

#### Editing the template

1. **Using the bidseditor**. While this is certainly not recommended for most cases, the easiest (quick and dirty) way to create a bidsmap template is to use the bidseditor GUI. 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). Note that you should first clear the attribute values (e.g. `EchoTime`) that vary across repeats of the same or similar acquisitions. With the GUI you can still use advanced features, such as regular expression patterns for the attribute values. You can also open the template bidsmap itself with the bidseditor and edit it directly. The main limitation of using the GUI is that the run items are simply appended to a bidsmap template, meaning that they are last in line (for that datatype) when the bidsmapper tries to find a matching run-item. Another (smaller) limitation is that with the GUI you cannot make usage of YAML anchors and references, yielding a less clearly formatted bidsmap that is harder to maintain. Both limitations are overcome when directly editing the template bidsmap yourself using a text editor (see next point).
2. **Using a text editor**. This is the most powerful way to create or modify a bidsmap template but requires more knowledge of YAML and more understanding of bidsmaps. To organise and empower your template you can take the DCCN template bidsmap (`[path_to_bidscoin]/heuristics/bidsmap_dccn.yaml`) as an example and work from there. If you open that template with a text editor, there are a few handy 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 maintanance 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-item. In that way, however, the subsequent runs that dereference (e.g. with `<<: *anatattributes_dicom`) this anchor (e.g. `&anatattributes_dicom`) will inherit only the keys and can inject their own values, as shown in the

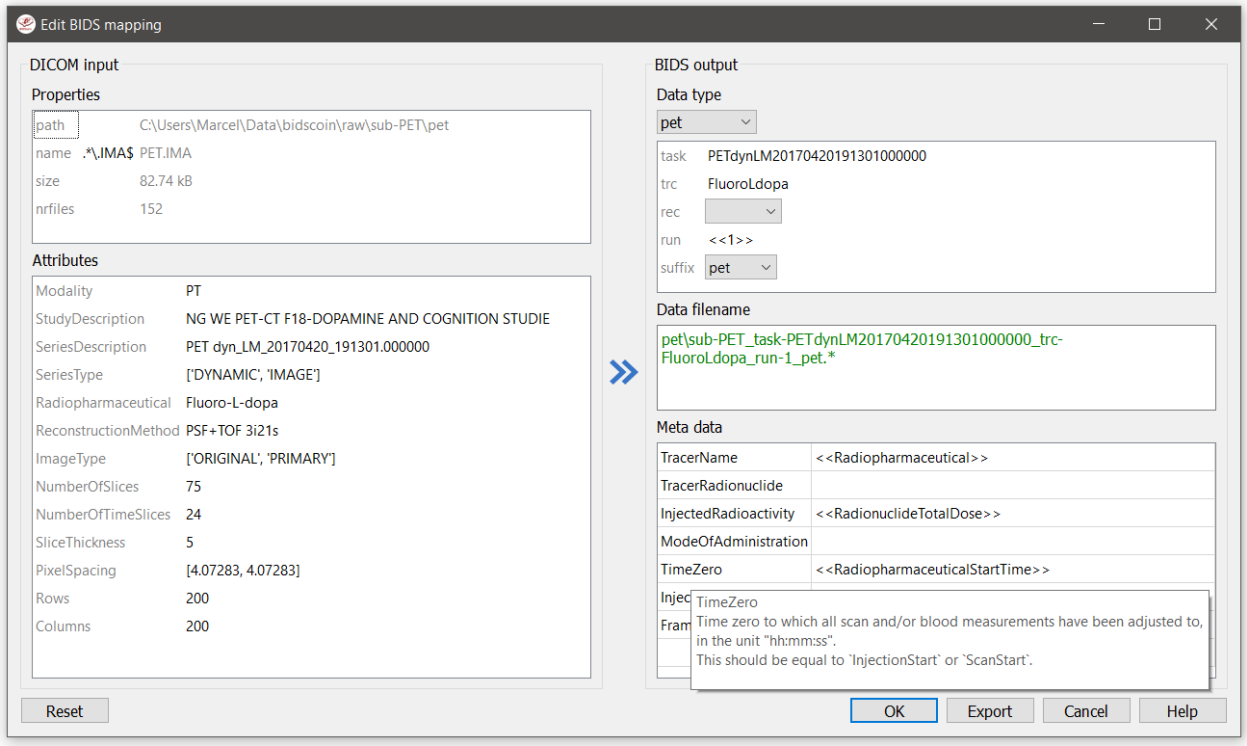



Fig. 7: The edit window with the option to export the customized mapping of run a item, and featuring filesystem matching and dynamic meta-data values

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 bidseditor GUI.

**Tip:**

- Run-items are matched from top to bottom. You can use this to your advantage by placing certain run-items above others
- The power of regular expressions is nearly unlimited, you can e.g. use negative look aheads to *not* match (exclude) certain strings
- Use more attributes for more selective run-item matching. For instance, to distinguish an equally named SBRef DWI scan from the normal DWI scans, you can add `DiffusionDirectionality: NONE` to your attribute dictionary

```
anat:       # ---------------------- All anatomical runs --------------------
- provenance: ~                 # The fullpath name of the DICOM file from which the␣
↪attributes are read. Serves also as a look-up key to find a run in the bidsmap
  filesystem: &fileattr         # This is an optional (stub) entry of filesystem␣
↪matching (could be added to any run-item)
    path: ~                     # File folder, e.g. ".*Parkinson.*" or ".
↪*(phantom|bottle).*"
    name: ~                     # File name, e.g. ".*fmap.*" or ".*(fmap|field.?
↪map|B0.?map).*"
    size: ~                     # File size, e.g. "2[4-6]\d MB" for matching files␣
↪between 240-269 MB
```

(continues on next page)

(continued from previous page)

```
    nrfiles: ~                  # Number of files in the folder that match the above␣
↪criteria, e.g. "5/d/d" for matching a number between 500-599
  attributes: &anat_dicomattr   # An empty / non-matching reference dictionary that␣
↪can be derefenced in other run-items of this data type
    Modality: ~
    ProtocolName: ~
    SeriesDescription: ~
    ImageType: ~
    SequenceName: ~
    SequenceVariant: ~
    ScanningSequence: ~
    MRAcquisitionType: ~
    SliceThickness: ~
    FlipAngle: ~
    EchoNumbers: ~
    EchoTime: ~
    RepetitionTime: ~
    PhaseEncodingDirection: ~
  bids: &anat_dicoment_nonparametric  # See: schema/datatypes/anat.yaml
    acq: <SeriesDescription>    # This will be expanded by the bidsmapper (so the␣
↪user can edit it)
    ce: ~
    rec: ~
    run: <<1>>                  # This will be updated during bidscoiner runtime (as␣
↪it depends on the already existing files)
    part: ['', 'mag', 'phase', 'real', 'imag', 0]
    suffix: T1w
  meta:                         # This is an optional entry for meta-data that will␣
↪be appended to the json sidecar files produced by dcm2niix
- provenance: ~
  filesystem:
    <<: *fileattr
    nrfiles: [1-3]/d/d          # Number of files in the folder that match the above␣
↪criteria, e.g. "5/d/d" for matching a number between 500-599
  attributes:
    <<: *anat_dicomattr
    ProtocolName: '(?i).*(MPRAGE|T1w).*'
    MRAcquisitionType: '3D'
  bids: *anat_dicoment_nonparametric
  meta:
    Comments: <<ImageComments>>    # This will be expanded during bidscoiner runtime␣
↪(as it may vary from session to session)
- provenance: ~
  attributes:
    <<: *anat_dicomattr
    ProtocolName: '(?i).*T2w.*'
    SequenceVariant: '[''SK'', ''SP'']'       # NB: Uses a yaml single-quote escape
  bids:
    <<: *anat_dicoment_nonparametric
    suffix: T2w
```

*Snippet derived from the bidsmap_dccn template, showing a 'DICOM' section with a void 'anat' run-item and two normal run-items that dereference from the void item*

### 2.7.2 Plugins

BIDScoin uses a flexible plugin architecture to map and convert your source data to BIDS. The bidsmapper and bidscoiner tools loop over the subjects/sessions in your source directory and then call the plugins listed in the bidsmap to do the actual work. As can be seen in the API code snippet below, the plugins can contain optional functions for interacting with their dataformat and for mapping and converting the source data to BIDS. See also the default `dcm2bidsmap` and `dcm2niix2bids` plugins for reference implementation.

**Note:** Plugins can be listed, installed and uninstalled using the central `bidscoin` command-line tool.

```
"""
This module contains placeholder code demonstrating the bidscoin plugin API, both for the bidsmapper and for
the bidscoiner. The functions in this module are called if the basename of this module (when located in the
plugins-folder; otherwise the full path must be provided) is listed in the bidsmap. The presence of the
plugin functions is optional but should be named:

- test:                 A test routine for the plugin + its bidsmap options. Can be called in the bidseditor
- is_sourcefile:        A routine to assess whether the file is of a valid dataformat for this plugin
- get_attribute:        A routine for reading an attribute from a sourcefile
- bidsmapper_plugin:    A routine that can be called by the bidsmapper to make a bidsmap of the source data
- bidscoiner_plugin:    A routine that can be called by the bidscoiner to convert the source data to bids
"""

import logging
from pathlib import Path

LOGGER = logging.getLogger(__name__)


def test(options: dict) -> bool:
    """
    This plugin function tests the working of the plugin + its bidsmap options

    :param options: A dictionary with the plugin options, e.g. taken from the bidsmap['Options']
    :return:        True if the test was successful
    """

    LOGGER.debug(f'This is a demo-plugin test routine, validating its working with options: {options}')

    return True


def is_sourcefile(file: Path) -> str:
    """
    This plugin function assesses whether a sourcefile is of a supported dataformat

    :param file:    The sourcefile that is assessed
```

(continues on next page)

(continued from previous page)

```
    :return:        The valid / supported dataformat of the sourcefile
    """

    if file.is_file():

        LOGGER.debug(f'This is a demo-plugin is_sourcefile routine, assessing whether
↪"{file}" has a valid dataformat')
        return 'dataformat'

    return ''



def get_attribute(dataformat: str, sourcefile: Path, attribute: str) -> str:
    """
    This plugin function reads attributes from the supported sourcefile

    :param dataformat:  The dataformat of the sourcefile, e.g. DICOM of PAR
    :param sourcefile:  The sourcefile from which key-value data needs to be read
    :param attribute:   The attribute key for which the value needs to be retrieved
    :return:            The retrieved attribute value
    """

    if dataformat in ('DICOM','PAR'):
        LOGGER.debug(f'This is a demo-plugin get_attribute routine, reading the
↪{dataformat} "{attribute}" attribute value from "{sourcefile}"')

    return ''


def bidsmapper_plugin(session: Path, bidsmap_new: dict, bidsmap_old: dict, template:␣
↪dict, store: dict) -> None:
    """
    All the logic to map the Philips PAR/XML fields onto bids labels go into this␣
↪plugin function. The function is
    expecte to update / append new runs to the bidsmap_new data structure. The␣
↪bidsmap options for this plugin can
    be found in:

    bidsmap_new/old['Options']['plugins']['README']

    See also the dcm2bidsmap plugin for reference implementation

    :param session:     The full-path name of the subject/session raw data source␣
↪folder
    :param bidsmap_new: The study bidsmap that we are building
    :param bidsmap_old: Full BIDS heuristics data structure, with all options, BIDS␣
↪labels and attributes, etc
    :param template:    The template bidsmap with the default heuristics
    :param store:       The paths of the source- and target-folder
    :return:
    """

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


def bidscoiner_plugin(session: Path, bidsmap: dict, bidsfolder: Path, personals: dict,
 ↪ subprefix: str, sesprefix: str) -> None:
```

(continues on next page)

(continued from previous page)

```
    """
    The plugin to convert the runs in the source folder and save them in the bids␣
↪folder. Each saved datafile should be
    accompanied with a json sidecar file. The bidsmap options for this plugin can be␣
↪found in:

    bidsmap_new/old['Options']['plugins']['README']

    See also the dcm2niix2bids plugin for reference implementation

    :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
    :param subprefix:   The prefix common for all source subject-folders
    :param sesprefix:   The prefix common for all source session-folders
    :return:            Nothing
    """

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

*The README plugin placeholder code*

## 2.8 Screenshots

### 2.8.1 The bidseditor

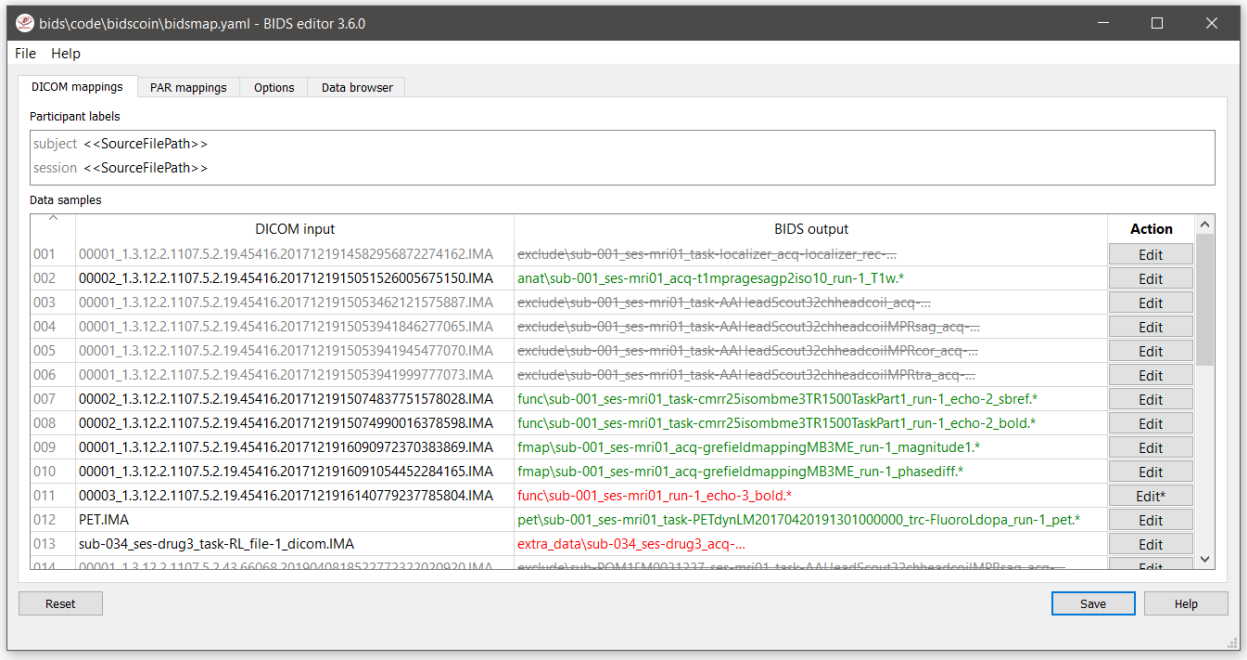



Fig. 8: The main window with an overview of all the bidsmap run items

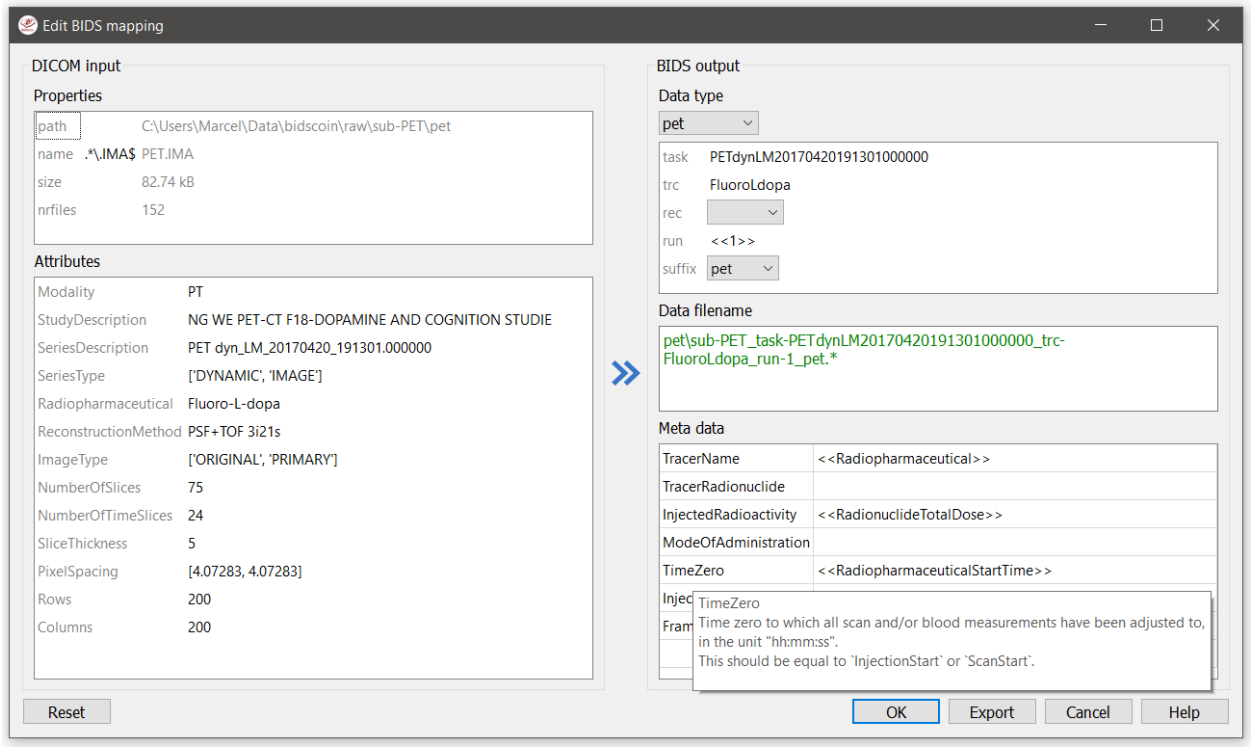



Fig. 9: The edit window with the option to export the customized mapping of run a item, and featuring filesystem matching and dynamic meta-data values

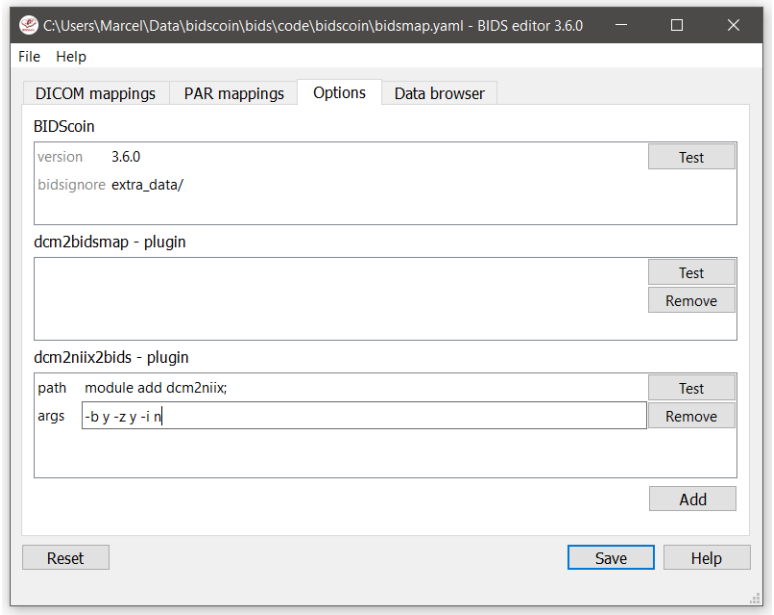



Fig. 10: The options window with BIDScoin settings

## 2.9 Demo and tutorial

### 2.9.1 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.

### 2.9.2 BIDScoin tutorial

1. **Data preparation.** Activate the bidscoin environment and create a tutorial playground folder in your home directory by executing these bash commands (users from outside the DCCN may have to adapt the first two commands to their environment):

```
$ module add bidscoin                # Load the DCCN bidscoin module with the PATH␣
↪settings and Anaconda environment
$ source activate /opt/bidscoin      # Activate the python virtual environment with␣
↪the BIDScoin python packages
$ bidscoin --download .              # Download the tutorial data (use a "." for the␣
↪current folder or adapt it to your needs)
$ cd bidscointutorial                # Go to the downloaded data (or provide the path␣
↪to the subfolders when calling the bidscoin tools)
```

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 (aka "runs"):

```
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 (hint: it's 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 study 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, edit the task and acquisition labels 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 user friendly, e.g. by naming the aquisition label simply `acq-mprage`.
- Add a search pattern to the `IntendedFor` field such that the first fieldmap will select your `Reward` runs and the second fieldmap your `Stop` runs (see the bidseditor fieldmap notes 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, go to the `Options` tab and change the `dcm2niix` settings to get non-zipped nifti output data (i.e. `*.nii` instead of `*.nii.gz`). Test the tool to see if it can run and, as a final step, save your bidsmap. You can always go back later to change any of your edits by running the bidseditor command line tool directly. Try that.

3. **BIDS coining.** The next step, converting the source data into a BIDS collection, is very simple to do (and can be repeated whenever new data has come in). To do this run the bidscoiner command-line 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 foldernames 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 echo-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 `derivatives` 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?

## 2.10 Changelog

All notable changes to this project will be documented in this file. The format is based on Keep a Changelog

### 2.10.1 3.7.0-dev

#### Added

- A BIDScoin installation test (`bidscoin -t`)
- A bidseditor button to save the Options to a (default) template bidsmap

#### Changed

- Plugins should now have a `is_sourcefile` and a `get_attribute` function

### 2.10.2 3.6.1 - 2021-05-20

#### Fixed

The bidscoiner no longer sometimes crashes when dcm2niix produces custom suffixes (e.g. for multi-echo data)

### 2.10.3 3.6.0 - 2021-05-13

#### Added

- Support for BIDS v1.6.0 (-> PET)
- Separate tabs for DICOM and PAR to edit all the mappings of mixed datasets in a single bidseditor session
- Run-item matching on filesystem properties, i.e. on the pathname, filename and filesize and nr of files in the folder. This can be used in conjunction with the (DICOM/PAR) attributes
- A meta-data dictionary that can be edited with the bidseditor and that will be added to the json sidecar files by the bidscoiner
- More user feedback in the GUI for new BIDS-compliancy checks on missing or invalid bids data
- A right-click menu option to remove a run-item from the bidsmap (advanced usage)
- The option to load a new bidsmap in the bidseditor
- Enable the user to edit json, yaml, tsv and other non-DICOM / non-PAR files with double-clicks in the data browser
- A central 'bidscoin' package function with various utilities, such as listing and installing BIDScoin plugins or executables
- Plugins can have their own 'test' routine that can be called from the bidseditor

#### Changed

- Using regular expressions instead of fnmatch to match (template bidsmap) attribute values. This makes the templates more powerful and flexible
- Moved the bidsmapping and bidscoining functionality to stand-alone plugins (changed API), making plugins a first-class BIDScoin citizen
- The plugins have moved to the bidsmap['Options'], where they have their own key-value options dictionary (changed API)

- Move IntendedFor field over to the new meta-data dictionary
- Renamed the `leave_out` datatype to `exclude`
- Re-introduced skipping hidden folders (hidden files are also skipped)
- Moved the 'pulltutorial' function over to the new 'bidscoin' function

#### Removed

- P7 and nifti support (it was never implemented anyhow)
- The option to edit new mappings on-the-fly in the bidsmapper (`-i 2`)

### 2.10.4 3.5.3 - 2021-04-13

#### Fixed

- Save non-standard fieldmaps in the derivative folder
- Add 'AcquisitionTime' to physio json-files and add the physio-files to the *_scans.tsv file

### 2.10.5 3.5.2 - 2021-03-21

#### Fixed:

- pypi upload

### 2.10.6 3.5.1 - 2021-03-21

#### Added

- BIDScoin version update checks

#### Fixed

- Speed optimizations
- Code clean-up
- More robust dcm2niix output handling

### 2.10.7 3.5 - 2021-03-08

A significant rewrite and evolution of BIDScoin!

#### Added

- Support for BIDS v1.5
- Support for Siemens advanced physiological logging data
- Improved GUI help tooltips and user feedback
- Improved feedback and control for invalid bidsnames
- Validation of run-items and bidsmaps against the BIDS schema

#### Changed

- Use the dccn template bidsmap as the default

#### Fixed

- Simplified and improved (hopefully) handling of fieldmaps

### 2.10.8 3.0.8 - 2020-09-28

#### Fixed

- Various minor bugs

### 2.10.9 3.0.6 - 2020-08-05

#### Fixed

- Minor but important bugfix in the setup :-)

### 2.10.10 3.0.5 - 2020-08-05

#### Added

- A download tool for tutorial data
- A tool for regenerating the participants.tsv file

#### Fixed

- Various bugs

### 2.10.11 3.0.4 - 2020-05-14

#### Added

- `Export` function in the bidseditor to allow for adding run items to existing (template) bidsmaps
- Support for Unix-shell style wildcards for matching run items in the bidsmap

#### Changed

- Improved DCCN example template bidsmap

#### Fixed

- Various minor bugs

### 2.10.12 3.0.3 - 2020-04-14

#### Fixed

- A small bugfix to properly handle appending dcm2niix suffices to the BIDS acq-label

### 2.10.13 3.0.2 - 2020-04-06

#### Fixed

- Special thanks to Thom Shaw, who was patient enough to keep testing untested bugfixes (#56) and helped making BIDScoin better :-)

### 2.10.14 3.0.1 - 2020-04-04

#### Added

- A 'provenance store' in the `bidsmapper` to fix a bug (#56 ) and allow for moving the bids-folder around
- Support for zipped/tarred DICOM directories

### 2.10.15 3.0 - 2020-04-01

A Significant rewrite to make BIDScoin more robust, user friendly and feature-rich :-)

#### Added

- First support for Philips PAR / REC data format
- A BIDS compliant defacing tool
- A BIDS compliant multi-echo combination tool
- Much improved documentation (https://bidscoin.readthedocs.io)

### 2.10.16 2.3.1 - 2019-09-12

#### Fixed

- a small but important bug that caused datasets without fieldmaps to crash (my test datasets all had fieldmaps :-))

### 2.10.17 2.3 - 2019-08-29

A lot of improvements have landed in 2.3, making it the best release of the 2-series by far!

#### Added

- The possibility to edit Participant labels
- Various tests and checks in Options to ensure creating good working bidsmaps / BIDS output data
- Upgraded compliance with bids v1.2.1
- The possibility to leave-out certain data types / runs

#### Changed

- A new workflow that is easier and more consistent
- Greatly improved graphical user interface and error/warning reporting
- Improved bidsmap_dccn template

#### Fixed

- Significant code refactoring to squash a number of important bugs and make the code more robust and maintainable

### 2.10.18 2.2 - 2019-07-11

#### Added

- Options tab to edit and test the bidscoin Options
- A leave-out option (to ignore runs / prevent them from showing up in the BIDS directory)
- A graphical interface to the bidsmapper
- Improved logging
- Improved the DICOM attribute *wildcard* feature

#### Changed

- New layout of the main and edit windows

#### Fixed

- Various bugfixes

### 2.10.19 2.1 - 2019-06-23

#### Added

- Editing of bidsmap Options

#### Fixed

- `IntendedFor` in fieldmap json sidecar files
- Code redundancy

### 2.10.20 2.0 - 2019-06-18

A major release and rewrite with important user-facing improvements

#### Added

- A shiny GUI :-)
- A new and much easier workflow

#### Fixed

- Various bugfixes

### 2.10.21 1.5 - 2019-03-06

#### Added

- Support for PET scans
- Support for DICOMDIR data
- Saving of template sidecar files in the bids output directory

#### Changed

- increased flexibility for renaming / reorganising the raw (input) data structure
- Added provenance data to the bidsmap/yaml files

#### Fixed

- various bugfixes

### 2.10.22 1.4 - 2018-10-22

#### Added

- Cross platform support
- Installation as a python module
- Improved version control
- Improved BIDS compliance

### 2.10.23 1.3 - 2018-09-28

#### Changed

- Refactored bidsmap naming

#### Fixed

- Various bugs

### 2.10.24 1.2 - 2018-09-14

#### Added

- Improved fieldmap support

#### Changed

- Yaml-syntax

### 2.10.25 1.0 - 2018-07-04

A first stable release of BIDScoin :-)

#### Added

- Support the conversion of organised sub/ses DICOM folders to BIDS

#### To do

- Add support for non-imaging data