BIDScoin

Release 3.7.0

Marcel Zwiers

Dec 24, 2021

CONTENTS

BIDScoin is a user friendly open-source Python application that converts ("coins") source-level (raw) neuroimaging data-sets to nifti / json / tsv data-sets that are organized according to the Brain Imaging Data Structure (BIDS) standard. Rather then depending on complex programmatic logic for source data-type identification, BIDScoin uses a mapping approach to identify and convert the source data types into BIDS data types. Different runs of source data are identified by their file system properties (e.g. file name or size) and by their attributes (e.g. `ProtocolName` from the DICOM header). Mapping information about how these runs should be converted to BIDS can be pre-specified (e.g. per site) by the user, allowing BIDScoin to make intelligent first suggestions on how to convert the data. While this command-line procedure exploits all information available on disk, BIDScoin presents a Graphical User Interface (GUI) for researchers to check and edit these mappings – bringing in the missing knowledge that often exists only in their heads.

BIDScoin requires no programming knowledge in order to use it, but users can use regular expression and plug-ins to further enhance BIDScoin's power and flexibilty, and deal with a wider range of source data formats.

BIDScoin is developed at the Donders Institute of the Radboud University.

CHAPTER

ONE

BIDSCOIN FUNCTIONALITY

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

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

  ```
  ** = Only Twix, SDAT/SPAR and P-file 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.
```

CHAPTER

TWO

NOTE:

The full BIDScoin documentation is hosted at Read the Docs

For citation and more information, see our BIDScoin publication **in Frontiers in Neuroinformatics** (doi: 10.3389/fninf.2021.770608)

Issues can be reported at Github

2.1 Installation

2.1.1 Python installation

BIDScoin is a Python 3 package and can be installed on Linux, MS Windows and on OS-X computers, as long as a Python interpreter (v3.8 or higher) is available on the system. On Linux and OS-X this is usually already the case, but MS Windows users may need to first install Python themselves. See e.g. this Python 3 distribution for instructions.

2.1.2 BIDScoin installation

To install BIDScoin on your system run one of the following commands in your command-line interface / shell (tip: you may want or need to install bidscoin in a virtual / conda Python environment):

```
$ pip install bidscoin                         # Use this when you want to convert␣
↪conventional MR imaging data with the dcm2niix2bids plugin
$ pip install bidscoin[spec2nii2bids]          # Use this when you want to convert MR␣
↪spectroscopy data with the spec2nii2bids plugin
$ pip install bidscoin[phys2bidscoin]          # Use this when you want to convert␣
↪physiological data with the phys2bidscoin plugin -- EXPERIMENTAL!
$ pip install bidscoin[deface]                 # Use this when you want to deface␣
↪anatomical MRI scans. NB: Requires FSL to be installed on your system
$ pip install bidscoin[deface,spec2nii2bids]   # Use this to install two extra␣
↪packages of interest
$ pip install bidscoin[all]                    # Use this to install all extra packages
```

These install commands can be run independently and will give you the latest stable release of BIDScoin and its plugins. Alternatively, if you need to use the very latest (development / unstable) version of the software, you can also install BIDScoin 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 folder named e.g. ‘bidscoin’ and run:

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

Updating BIDScoin

Run your pip install command as before with the additional `--upgrade` option, e.g.:

```
$ 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 your template bidsmap (so make a back-up of it before you upgrade)

2.1.3 Dcm2niix installation

Unfortunately the pip installer can only install Python software and the default 'dcm2niix2bids' plugin relies on an external application named dcm2niix to convert DICOM and PAR/REC source data to nifti. To make use of the dcm2niix2bids plugin you should therefore download and install dcm2niix yourself according to the instructions. When done, make sure that the command to run the dcm2niix executable is set correctly in the Options section in your bidsmap. This can be done in two ways:

1. Open your template bidsmap with a text editor and adjust the settings as needed. The default template bidsmap is located in the [path_to_bidscoin]/heuristics subfolder – see the output of `bidscoin -t` for the fullpath location on your system.
2. Go to the Options tab the first time the BIDScoin GUI is launched and adjust the settings as needed. Then click the [Set as default] button to save the settings to your default template bidsmap.

2.1.4 Testing BIDScoin

You can run the 'bidscoin' utility to test the installation of your BIDScoin installation and settings:

```
$ bidscoin -t                          # Test with the default template bidsmap
$ bidscoin -t my_template_bidsmap      # Test with your custom template bidsmap
```

2.1.5 Using a singularity container

An alternative for installing Python, BIDScoin and it's dependencies yourself is to execute BIDScoin commands using a Singularity image. Read Singularity documentation for installation and usage instructions.

The current image includes:

- Debian stable,
- the latest version of dcm2niix,
- the latest stable release of BIDScoin and its spec2nii2bids and phys2bidscoin plugins.

Dependencies:

- Debian (or Debian-like, e.g., Ubuntu) host system,

- debootstrap package.

Building the image

Execute the following command to build the BIDScoin image.

```
$ sudo singularity build bidscoin.sif singularity.def
```

Run BIDScoin tools from the image

Execute BIDScoin tool using the following command:

```
$ singularity exec bidscoin.sif <bidscoin_tool> <bidscoin_tool_args>
```

Where `<bidscoin_tool>` is a BIDScoin tool (e.g., `bidsmapper`, `bidscoiner`, `dicomsort`) and `<bidscoin_tool_args>` are the tool's arguments.

If your data doesn't reside in home folder, add `--bind` Singularity argument which maps a folder from the host system to one inside the Singularity container.

```
$ singularity exec bidscoin.sif --bind <host_dir>:<container_dir> <bidscoin_tool>
↪<bidscoin_tool_args>
```

For example:

```
$ singularity exec --bind /my/data:/mnt bidscoin.sif bidscoiner /my/data/source /my/data/
↪bids
```

Tip: Since there is no fixed entry point to the container, you can also use it to execute dcm2niix.

Latest develop release

To install the latest develop realease of BIDScoin, substitute

```
pip3 install bidscoin
```

with

```
pip3 install --upgrade git+https://github.com/Donders-Institute/bidscoin
```

in the definition `singularity.def` file.

Speed up building the image

To speed up building the Singularity image, you can change the `apt` servers to download the packages from a location closer to you. For example, add the following line as the first command in the `%post` section of `singularity.def` file to download the packages from Austria (*at*).

```
echo 'deb http://ftp.at.debian.org/debian stable main' > /etc/apt/sources.list
```

Troubleshooting

The image didn't work after copying it to a CentOS 7 host system. The problem was kernel version older than 3.15. A working fix is to add the following line at the end of `%post` section of `singularity.def` file.

```
strip --remove-section=.note.ABI-tag /usr/lib/x86_64-linux-gnu/libQt5Core.so.5
```

The fix comes from these resources:

- (Answer #3) https://answers.launchpad.net/yade/+question/696260/
- https://github.com/wkhtmltopdf/wkhtmltopdf/issues/4497
- https://stackoverflow.com/questions/58912268/singularity-container-python-pytorch-why-does-import-torch-work-on-arch-l

2.2 Data preparation

2.2.1 Required source data structure

BIDScoin requires that the source data repository is organized according to a `subject/[session]/data` structure (the `session` subfolder is optional). The data folder can be structured in various ways, as illustrated by the following examples:

1. **A 'seriesfolder' organization**. The data folder is organised in multiple series subfolders, each of which that contains a single data type that is 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
```

(continues on next page)

(continued from previous page)

```
|       |   |-- 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**. The data folder contains a DICOMDIR file and multiple subfolders (NB: single DICOMDIR files containing multiple subjects or sessions are not supported). 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 the data folder contains all the DICOM files of all the different Series without any subfolders. 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
```

(continues on next page)

(continued from previous page)

```
|       [..]
|
|-- 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 contained in the data folder (without subfolders). 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 (i.e. the bidsmapper and the bidscoiner) will automatically run *dicomsort* in a temporary folder to sort them in seriesfolders. Depending on the data and file system, repeatedly unzipping data in the workflow may come with a significant processing 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 (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.py [-h] [-i SUBPREFIX] [-j SESPREFIX] [-f FOLDERSCHEME] [-n NAMESCHEME]␣
↪[-p PATTERN] [-d]
                    dicomsource

Sorts and / or renames DICOM files into local subfolders, e.g. with 3-digit SeriesNumber-
↪SeriesDescription
folder names (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 FOLDERSCHEME, --folderscheme FOLDERSCHEME
                        Naming scheme for the sorted DICOM Series subfolders. Follows␣
↪the Python string
                        formatting syntax with DICOM field names in curly bracers with␣
↪an optional
                        number of digits for numeric fields. Sorting in subfolders is␣
↪skipped when an
                        empty folderscheme is given (but note that renaming the␣
↪filenames can still be
                        performed) (default: {SeriesNumber:03d}-{SeriesDescription})
  -n NAMESCHEME, --namescheme NAMESCHEME
                        Optional naming scheme that can be provided to rename the DICOM␣
↪files. Follows
                        the Python string formatting syntax with DICOM field names in␣
↪curly bracers with
                        an optional number of digits for numeric fields. Use "
↪{PatientName}_{SeriesNumbe
                        r:03d}_{SeriesDescription}_{AcquisitionNumber:05d}_
↪{InstanceNumber:05d}.IMA" for
```

(continues on next page)

(continued from previous page)

```
                        the default names at DCCN (default: None)
  -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 sub-011/ses-mri01
  dicomsort sub-011/ses-mri01/DICOMDIR -n {AcquisitionNumber:05d}_{InstanceNumber:05d}.
↪dcm
  dicomsort /project/3022026.01/raw --subprefix sub
  dicomsort /project/3022026.01/raw --subprefix sub-01 --sesprefix ses
```

rawmapper

Another command-line utility that can be helpful in organizing your source data is `rawmapper`. This utility can show you an overview (map) of all the values of DICOM-attributes of interest in your data-set and, optionally, used to rename your source data sub-folders. The latter option 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.py [-h] [-s SESSIONS [SESSIONS ...]] [-f FIELD [FIELD ...]] [-w␣
↪WILDCARD]
                    [-o OUTFOLDER] [-r] [-n SUBPREFIX] [-m SESPREFIX] [-d]
                    sourcefolder

Maps out the values of a dicom attribute 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 at the scanner console (i.e. this data is stored in␣
↪the dicom
attribute named 'PatientComments')

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)
  -f FIELD [FIELD ...], --field FIELD [FIELD ...]
                        The fieldname(s) of the dicom attribute(s) used to rename or map␣
↪the
```

(continues on next page)

(continued from previous page)

```
                        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-)
  -d, --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.

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 shell commands, e.g.:

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

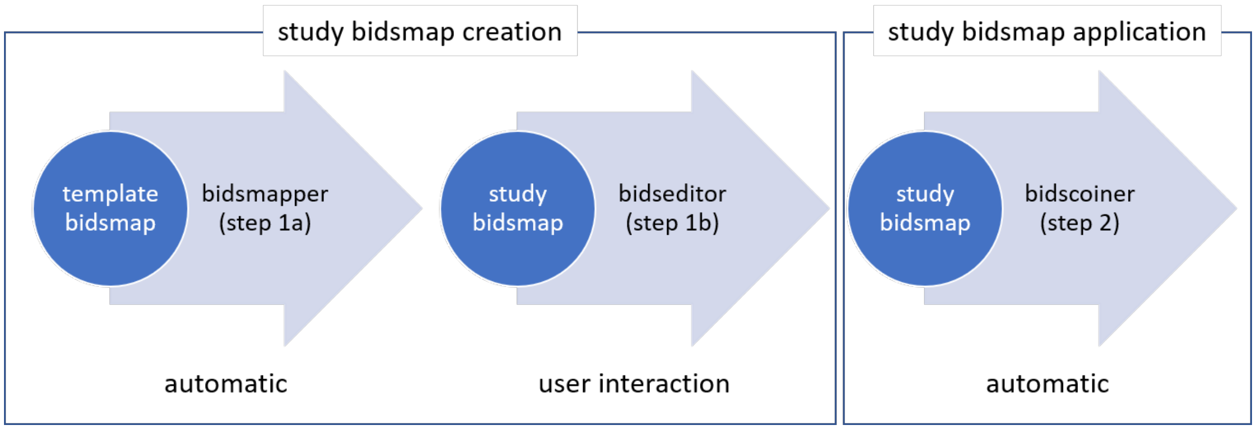


Fig. 1: Creation and application of a study bidsmap

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 run-items in the template bidsmap. Once a match is found, a mapping to␣
↪BIDS
output data types is made and the run-item is added to the study bidsmap. 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
```

(continues on next page)

(continued from previous page)

```
                        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
  -p PLUGINS, --plugins PLUGINS
                        List of plugins to be used (with default options, overrules the␣
↪plugin list
                        in the study/template bidsmaps)
  -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)
  -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_dccn
```

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

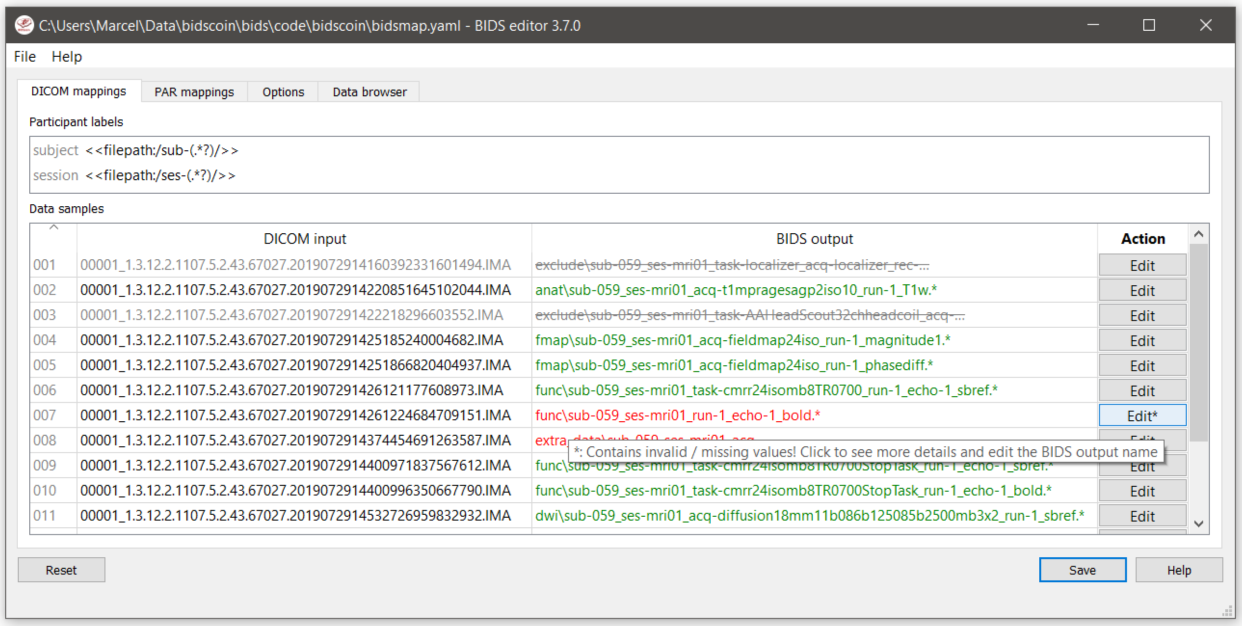
Tip: The default template bidsmap (`-t bidsmap_dccn`) is customized for acquisitions at the DCCN. If this bidsmap is not working well for you, consider adapting it to your needs so that the bidsmapper can recognize more of your scans and automatically map them to BIDS the way you prefer.

2.3.2 Step 1b: Running the bidseditor

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

This application 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 run-items 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-'
  -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
```

Main window

As shown below, the main window of the bidseditor opens with separate data mapping tabs for each data format that is present in the bidsmap (here `DICOM mappings` and `PAR mappings`). The data mapping tabs consist of a `Participant labels` table and a `Data samples` table. By default, the participant table contains dynamic <<`filepath:regexp`>> property values, which are used to extract the subject and session labels from the path of the source data during bidscoiner runtime. Alternatively, you can put a dynamic attribute value there (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.


C:\Users\Marcel\Data\bidscoin\bids\code\bidscoin\bidsmap.yaml - BIDS editor 3.7.0

Participant labels

subject <<filepath:/sub-(.*?)/>>
session <<filepath:/ses-(.*?)/>>

Data samples

	DICOM input	BIDS output	Action
001	00001_1.3.12.2.1107.5.2.43.67027.2019072914160392331601494.IMA	~~exclude\sub-059_ses-mri01_task-localizer_acq-localizer_rec-...~~	Edit
002	00001_1.3.12.2.1107.5.2.43.67027.2019072914220851645102044.IMA	anat\sub-059_ses-mri01_acq-t1mpragesagp2iso10_run-1_T1w.*	Edit
003	00001_1.3.12.2.1107.5.2.43.67027.2019072914222182966603552.IMA	~~exclude\sub-059_ses-mri01_task-AAHeadScout32chheadcoil_acq-...~~	Edit
004	00001_1.3.12.2.1107.5.2.43.67027.2019072914251852400046820.IMA	fmap\sub-059_ses-mri01_acq-fieldmap24iso_run-1_magnitude1.*	Edit
005	00001_1.3.12.2.1107.5.2.43.67027.2019072914251866820404937.IMA	fmap\sub-059_ses-mri01_acq-fieldmap24iso_run-1_phasediff.*	Edit
006	00001_1.3.12.2.1107.5.2.43.67027.2019072914261211776089731.IMA	func\sub-059_ses-mri01_task-cmrr24isomb8TR0700_run-1_echo-1_sbref.*	Edit
007	00001_1.3.12.2.1107.5.2.43.67027.2019072914261224684709151.IMA	func\sub-059_ses-mri01_run-1_echo-1_bold.*	Edit*
008	00001_1.3.12.2.1107.5.2.43.67027.2019072914374454691263587.IMA	extra_data\sub-059_ses-mri01_acq	Edit
009	00001_1.3.12.2.1107.5.2.43.67027.2019072914400971837567612.IMA	func\sub-059_ses-mri01_task-cmrr24isomb8TR0700StopTask_run-1_echo-1_sbref.*	Edit
010	00001_1.3.12.2.1107.5.2.43.67027.2019072914400996350667790.IMA	func\sub-059_ses-mri01_task-cmrr24isomb8TR0700StopTask_run-1_echo-1_bold.*	Edit
011	00001_1.3.12.2.1107.5.2.43.67027.2019072914532726959832932.IMA	dwi\sub-059_ses-mri01_acq-diffusion18mm11b086b125085b2500mb3x2_run-1_sbref.*	Edit

*: Contains invalid / missing values! Click to see more details and edit the BIDS output name

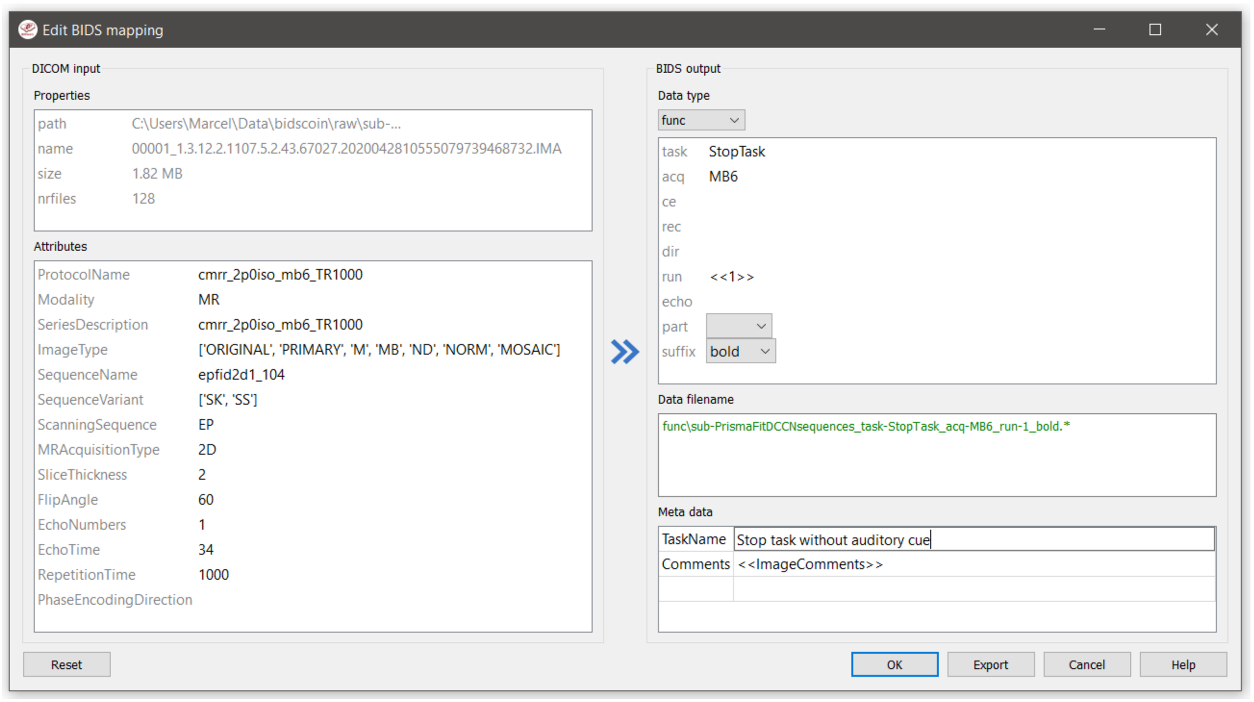
Fig. 2: The main window with the `DICOM mappings` and `PAR mappings` tabs, an `Options` tab and a `Data browser` tab. The selected `DICOM mappings` tab shows an overview of how DICOM source data types (left) are mapped to BIDS output data (right). The BIDScoin settings used for this study can be adjusted in the Options tab and the `Data browser` tab can be used to inspect the source data structure.

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 new window, as shown below. In this new 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. You should first make sure the BIDS `Data type` (drop down menu) and its `suffix` label (drop down menu) are set correctly, and then you should edit the (automatically generated) BIDS values that you think are not optimal or incorrect (double-click the cell). 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.



DICOM input

Properties

path	C:\Users\Marcel\Data\bidscoin\raw\sub-...
name	00001_1.3.12.2.1107.5.2.43.67027.2020042810555079739468732.IMA
size	1.82 MB
nrfiles	128

Attributes

ProtocolName	cmrr_2p0iso_mb6_TR1000
Modality	MR
SeriesDescription	cmrr_2p0iso_mb6_TR1000
ImageType	['ORIGINAL', 'PRIMARY', 'M', 'MB', 'ND', 'NORM', 'MOSAIC']
SequenceName	epfid2d1_104
SequenceVariant	['SK', 'SS']
ScanningSequence	EP
MRAcquisitionType	2D
SliceThickness	2
FlipAngle	60
EchoNumbers	1
EchoTime	34
RepetitionTime	1000
PhaseEncodingDirection	

BIDS output

Data type

func

task	StopTask
acq	MB6
ce	
rec	
dir	
run	<<1>>
echo	
part	
suffix	bold

Data filename

func\sub-PrismaFitDCCNsequences_task-StopTask_acq-MB6_run-1_bold.*

Meta data

TaskName	Stop task without auditory cue
Comments	<<ImageComments>>

Fig. 3: The edit window for customizing a bidsmap run item, featuring file name matching (.*.IMA) and dynamic metadata values (e.g. `TimeZero`). BIDS values that are restricted to a limited set are presented with a drop-down menu (here the `Data type`, the `rec` and the `suffix` value).

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 (step 2). Note that re-running the bidsmapper or bidseditor is always a safe thing to do since these tools will re-use the existing bidsmap yaml-file and will not delete or write anything to disk except to the 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 association 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!

If the run-index or any other value is not what you want it to be, try using dynamic values, as explained in the special bidsmap features section

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
                        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:
```

(continues on next page)

(continued from previous page)

```
                        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.

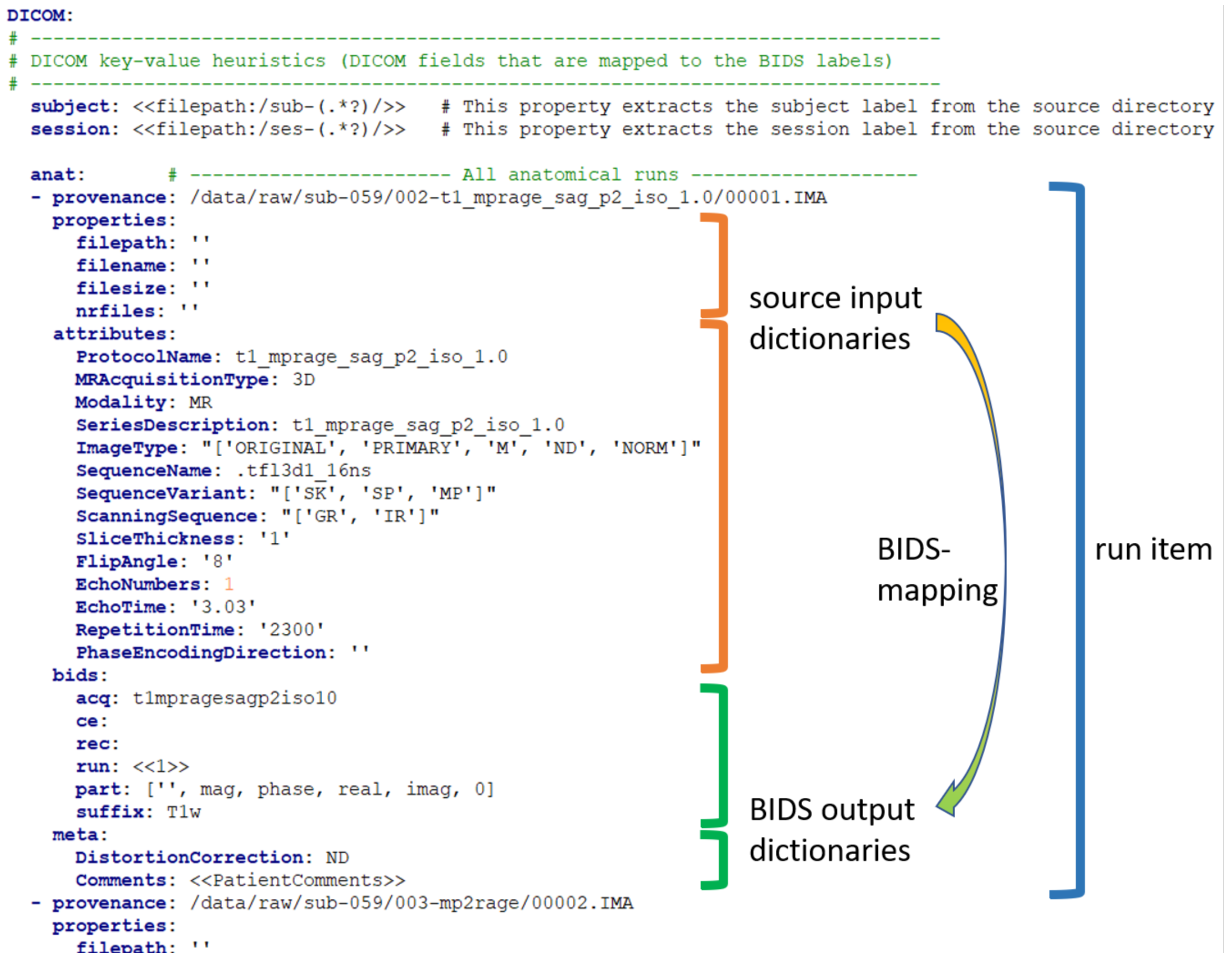
2.4 The bidsmap explained

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 types (e.g. a T1w- or a T2w-scan) should be converted to BIDS data types. As illustrated in the figure below (but see also the screenshot of the edit window), run-items consist of a 'provenance' field and a 'properties', 'attributes', 'bids' and a 'meta' dictionary (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 conversion process, e.g. in case of encountering unexpected results
2. The properties dictionary contains file system properties of the data sample, i.e. the file path, the file name, the file size on disk and the number of files in the containing folder. 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 of which a minimal subset 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 are encouraged to be edited by the user
5. The meta dictionary contains custom key-value pairs that are added to the json sidecar file by the BIDScoin plugins. Meta data may well vary from session to session, hence this dictionary often contains dynamic attribute values that are evaluated during bidscoiner runtime (see the *special features* below)

In sum, a run-item contains a single bids-mapping, which links the input dictionaries (2) and (3) to the output dictionaries (4) and (5).



```
DICOM:
# --------------------------------------------------------------------------------
# DICOM key-value heuristics (DICOM fields that are mapped to the BIDS labels)
# --------------------------------------------------------------------------------
  subject: <<filepath:/sub-(.*?)/>>   # This property extracts the subject label from the source directory
  session: <<filepath:/ses-(.*?)/>>   # This property extracts the session label from the source directory

  anat:       # ----------------------- All anatomical runs --------------------
  - provenance: /data/raw/sub-059/002-t1_mprage_sag_p2_iso_1.0/00001.IMA
    properties:
      filepath: ''
      filename: ''
      filesize: ''
      nrfiles: ''
    attributes:
      ProtocolName: t1_mprage_sag_p2_iso_1.0
      MRAcquisitionType: 3D
      Modality: MR
      SeriesDescription: t1_mprage_sag_p2_iso_1.0
      ImageType: "['ORIGINAL', 'PRIMARY', 'M', 'ND', 'NORM']"
      SequenceName: .tfl3d1_16ns
      SequenceVariant: "['SK', 'SP', 'MP']"
      ScanningSequence: "['GR', 'IR']"
      SliceThickness: '1'
      FlipAngle: '8'
      EchoNumbers: 1
      EchoTime: '3.03'
      RepetitionTime: '2300'
      PhaseEncodingDirection: ''
    bids:
      acq: t1mpragesagp2iso10
      ce:
      rec:
      run: <<1>>
      part: ['', mag, phase, real, imag, 0]
      suffix: T1w
    meta:
      DistortionCorrection: ND
      Comments: <<PatientComments>>
  - provenance: /data/raw/sub-059/003-mp2rage/00002.IMA
    properties:
      filepath: ''
```

Fig. 4: A snippet of study bidsmap in YAML format. The bidsmap contains separate sections for each source data format (here 'DICOM') and sub-sections for the BIDS datatypes (here 'anat'). The arrow illustrates how the 'properties' and 'attributes' input dictionaries are mapped onto the 'bids' and 'meta' output dictionaries. Note that the 'part' value in the bids dictionary is a list, which is presented in the bidseditor GUI as a drop-down menu (with the first empty item being selected). Also note the special double bracket dynamic values (<<..>>), which are explained *below*.

At the root level, a bidsmap is hierarchically organised in data format sections, such as 'DICOM' and 'PAR', which in turn contain subsections for the 'participant_label' and 'session_label', subsections 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 contain the actual run-items. Next to the data format sections there is a general 'Options' section, that accommodates BIDScoin and plugin settings.

When BIDScoin routines process source data, they will scan the entire repository and take samples of the data and compare them with the run-items in the bidsmap until they come across a run-item of which all (non-empty) properties and attribute values match with the values extracted from the data sample at hand. At that point a bidsmapping is established. 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.

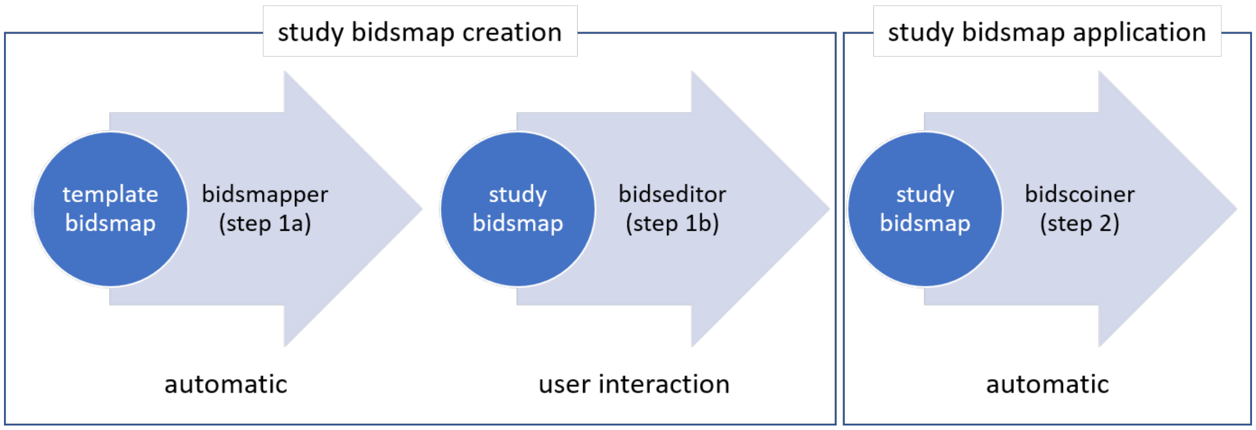


Fig. 5: Creation and application of a study bidsmap

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:

- **Run-item matching**. Source property and attribute values of run-items in a bidsmap are interpreted as regular expression patterns when they are matched with your source data samples. For instance, a key-value pair of an attribute dictionary in your template bidsmap could be `{ProtocolName: .*(mprage|T1w).*}`, which would test if the extracted attribute string for 'ProtocolName' from the DICOM header of a data sample contains either a 'mprage' or a 'T1w' substring. More precisely, the Python expression that is evaluated is: `match = re.fullmatch('.*(mprage|T1w).*', 't1_mprage_sag_p2_iso_1.0')` if the ProtocolName of the data sample is 't1_mprage_sag_p2_iso_1.0'.
- **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 single or double pointy brackets. In case of single pointy brackets the bids value will be replaced during bidsmapper, bidseditor and bidscoiner runtime by the value of the source attribute or property of the data sample at hand. It is also possible to then extract a substring from the source string by adding a colon-separated regular expression to the bids value. For instance the two dynamic values in `{acq: <MRAcquisitionType>Demo<SeriesDescription:t1_(.*?)_sag>}` will be replaced by `{acq: 3DDemoMPRAGE}` if the 'MRAcquisitionType' of the data sample is '3D' and 'SeriesDescription' is 't1_MPRAGE_sag_p2_iso_1.0'. More precisely, the Python expression that is evaluated for the second dynamic 'SeriesDescription' value is: `substring = re.findall('t1_(.*?)_sag',`

't1_mprage_sag_p2_iso_1.0'). If dynamic values are enclosed with double pointy brackets, the only difference is that they will be replaced only during bidscoiner runtime – this is useful for bids values that are subject/session dependent. Double bracket dynamic values can for instance be used to add DICOM meta data that is not saved by default in the json sidecar files, such as <<ImageComments>> or <<RadionuclideTotalDose>>. Another example is the extraction of the subject and session label. For instance, you can use <<filepath:/sub-(.*?)/>> to extract '003' (i.e. the shortest string between /sub- and /) if the data for that subject is in /data/raw/sub-003/ses-01. Alternatively, if the subject label is encoded in the DICOM PatientName as e.g. ID_003_anon, then <<PatientName:ID_(.*?)_>> would likewise extract '003'. To test out dynamic values (either with or without appended regular expressions), you can handily enter them in the bidseditor within single brackets to instantly obtain their resulting value.

- **Run-index**. Dynamic values can handle many use cases and can be used throughout BIDScoin. Yet there are two exceptions that cannot always be handled directly with dynamic values. The first exception is the 'run'-index in the bids output dictionary, since this index number cannot usually be determined from the data file alone. In that case, if the run-index is a dynamic number (e.g. {run: <<1>>}) and another output file with that run-index already exists, then during bidscoiner runtime this number will be incremented in compliance with the BIDS standard (e.g. to {run: 2}). If the run index is encoded in the header or filename, then the index can unambiguously be extracted using dynamic values. For instance, using {run: <<ProtocolName:run-(.*?)_>>} will give {run: 3} if the DICOM ProtocolName is t1_mprage_sag_run-3_iso_1.0.
- **IntendedFor**. The other exception not covered by dynamic values is the 'IntendedFor' value in the meta dictionary of fieldmaps. The IntendedFor value is a list of associated output files that you can specify within a dynamic value using Unix shell-style wildcards. In that way, the bidscoiner will lookup these images on disk using the Python glob(*dynamic_value*) expression. For instance, using {IntendedFor: <<task>>} will select all functional runs in the BIDS subject[/session] folder (since these runs always have 'task' in their filename), and using {IntendedFor: <<Stop*Go><Reward>>} will select all 'Stop1Go'-, 'Stop2Go'- and 'Reward'-runs.
- **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).

Tip: In addition to DICOM attribute names, the more advanced / unambiguous pydicom-style tag numbers can also be used for indexing a DICOM header. For instance, the PatientName, 0x00100010, 0x10,0x10, (0x10, 0x10), and (0010, 0010) index keys are all equivalent.

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: echocombine [-h] [-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]] [-o OUTPUT]
                   [-a {PAID,TE,average}] [-w [WEIGHTS [WEIGHTS ...]]] [-f]
                   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:
  bidsfolder            The bids-directory with the (multi-echo) subject data
  pattern               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.
                        '*task-*echo-1*'

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)
  -o OUTPUT, --output OUTPUT
                        A string that determines where the output is saved. It can be␣
↪the name of a BIDS
                        datatype folder, such as 'func', or of the derivatives folder, i.
↪e.
                        'derivatives'. If output = [the name of the input datatype␣
↪folder] then the
                        original echo images are replaced by one combined image. If␣
↪output is left empty
```

(continues on next page)

(continued from previous page)

```
                        then the combined image is saved in the input datatype folder␣
↪and the original
                        echo images are moved to the extra_data folder (default: )
  -a {PAID,TE,average}, --algorithm {PAID,TE,average}
                        Combination algorithm (default: TE)
  -w [WEIGHTS [WEIGHTS ...]], --weights [WEIGHTS [WEIGHTS ...]]
                        Weights for each echo (default: None)
  -f, --force           If this flag is given subjects will be processed, regardless of␣
↪existing target
                        files already exist. Otherwise the echo-combination will be␣
↪skipped (default:
                        False)

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 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 [-h] [-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]] [-o OUTPUT] [-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

For multi-echo data see `medeface`

positional arguments:
  bidsfolder            The bids-directory with the 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 OUTPUT, --output OUTPUT
                        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": ""}'
```

Multi-echo defacing

This utility is very similar to the *deface* utility above, except that it can handle multi-echo data.

```
usage: medeface [-h] [-m MASKPATTERN] [-p PARTICIPANT_LABEL [PARTICIPANT_LABEL ...]] [-o␣
↪OUTPUT] [-c]
                [-n NATIVESPEC] [-a ARGS]
                bidsfolder pattern

A wrapper around the 'pydeface' defacing tool (https://github.com/poldracklab/pydeface)␣
↪that computes
a defacing mask on a (temporary) echo-combined image and then applies it to each␣
↪individual echo-image.

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

For single-echo data see `deface`

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/*_T2starw*'

optional arguments:
  -h, --help            show this help message and exit
```

(continues on next page)

(continued from previous page)

```
  -m MASKPATTERN, --maskpattern MASKPATTERN
                        Globlike search pattern (relative to the subject/session folder)␣
↪to select the
                        images from which the defacemask is computed, e.g. 'anat/*_part-
↪mag_*_T2starw*'.
                        If not given then 'pattern' is used (default: None)
  -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)
  -o OUTPUT, --output OUTPUT
                        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:
  medeface /project/3017065.01/bids anat/*_T1w*
  medeface /project/3017065.01/bids anat/*_T1w* -p 001 003 -o derivatives
  medeface /project/3017065.01/bids anat/*_T1w* -c -n "-l walltime=00:60:00,mem=4gb"
  medeface /project/3017065.01/bids anat/*acq-GRE* -m anat/*acq-GRE*magnitude*"
  medeface /project/3017065.01/bids anat/*_FLAIR* -a '{"cost": "corratio", "verbose": ""}
↪'
```

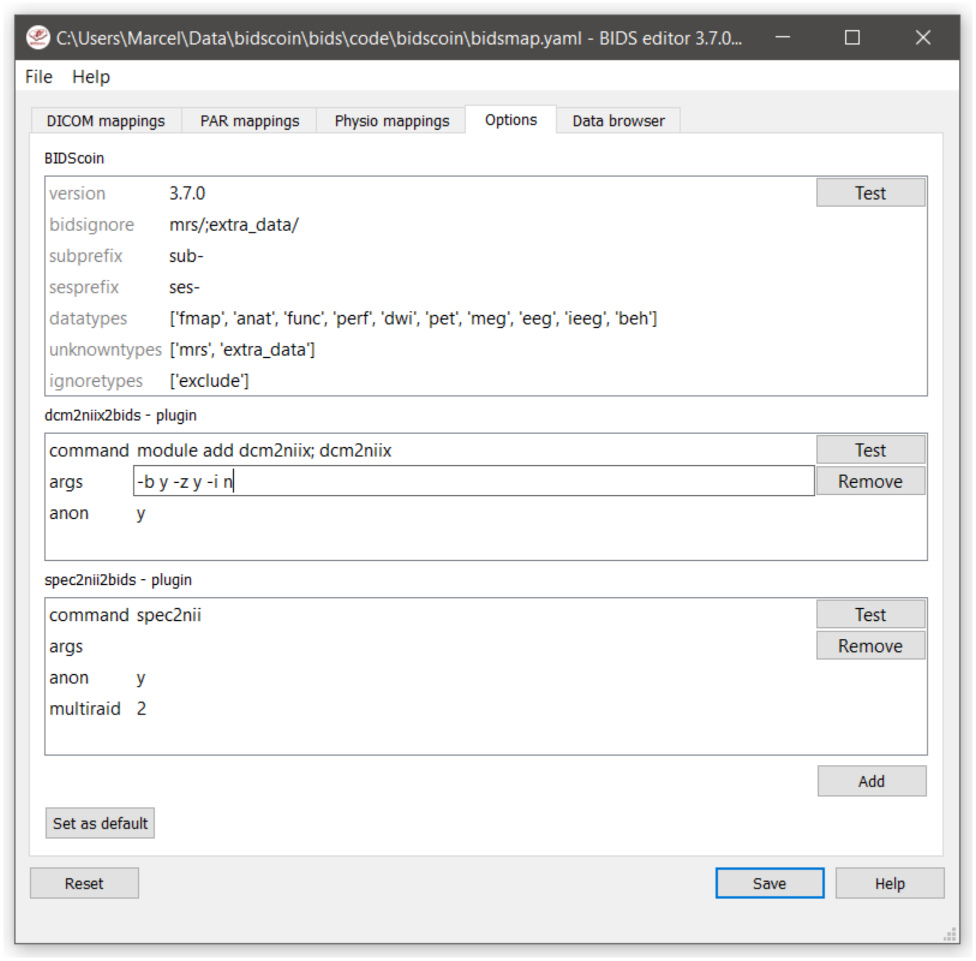
2.5.3 BIDS validation

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, set as default in the template bidsmap. There are separate settings for BIDScoin and for the individual plugins that can be edited by double clicking the corresponding fields. Installed plugins can be removed or added to extend BIDScoin's functionality.



BIDScoin

version	3.7.0
bidsignore	mrs/;extra_data/
subprefix	sub-
sesprefix	ses-
datatypes	['fmap', 'anat', 'func', 'perf', 'dwi', 'pet', 'meg', 'eeg', 'ieeg', 'beh']
unknowntypes	['mrs', 'extra_data']
ignoretypes	['exclude']

dcm2niix2bids - plugin

command	module add dcm2niix; dcm2niix
args	-b y -z y -i n
anon	y

spec2nii2bids - plugin

command	spec2nii
args	
anon	y
multiraid	2

Fig. 6: The bidseditor options window with the different settings for BIDScoin and its plugins. The user can manage the plugins that will be used with the [Add] and [Remove] buttons, and save the current options to the template bidsmap by using the [Set as default] button.

2.6.1 BIDScoin

These setting can be used by all the BIDScoin tools:

- `version`: Used to check for version conflicts between the installed version (see ../bidscoin/version.txt) and the version that was used to create the bidsmap, or between the installed version and the latest online version.
- `bidsignore`: Semicolon-separated list of (non-BIDS) datatypes that you want to include but that do not pass a BIDS validation test. These files are added to the *.bidsignore* file. Example: `bidsignore: extra_data/;rTMS/;myfile.txt;yourfile.csv`

- `subprefix`: The prefix before the subject label in the source data folder, e.g. 'patient-' if the source data is in `raw/patient-001/ses-01/..`
- `sesprefix`: Idem for the session label
- `datatypes`: Datatypes that are converted to BIDS
- `unknowntypes`: Datatypes that are not part of BIDS but that are converted to a BIDS-like entries in the BIDS folder
- `ignoretypes`: Datatypes that are excluded / not converted""""

The core working of BIDScoin can be tested by clicking the [Test] button and inspection of the terminal output.

2.6.2 dcm2niix2bids - plugin

The dcm2niix2bids plugin is 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:

- `command`: Command to run dcm2niix from the terminal, such as:
  - `dcm2niix` (if the executable is already present on your path)
  - `module add dcm2niix/v1.0.20210317; dcm2niix` (if you use a module system)
  - `PATH=/opt/dcm2niix/bin:$PATH; dcm2niix` (prepend the executable to your path)
  - `/opt/dcm2niix/bin/dcm2niix` (specify the fullpath to the executable)
  - `"C:\Program Files\dcm2niix\dcm2niix.exe"` (use quotes to deal with whitespaces in your fullpath)
- `args`: Argument string that is passed as input to dcm2niix to customize its behavior, e.g. `-z n -i y` for ignoring derived data and having uncompressed output data.
- `anon`: Set this anonymization flag to 'y' to round off age and to discard acquisition date from the meta data

To test the proper working of dcm2niix click [Test] and see the terminal output for more helptext on its input arguments

Tip:

- Use the [Set as default] button to put your custom dcm2niix command in your template bidsmap so that you don't have to adjust it anymore for every new study
- SPM users may want to use '-z n', which produces unzipped nifti's

2.6.3 spec2nii2bids - plugin

The spec2nii2bids plugin is a bidscoiner plugin that converts Twix, SPAR/SDAT and P-file spectroscopy source data to BIDS-valid nifti- and json sidecar files. This plugin relies on spec2nii, for which you can set the following options:

- `command`: Command to run spec2nii, such as:
  - `dcm2niix` (normal usage, i.e. the executable is already present on your path)
  - `module add spec2nii; spec2nii` (if you use a module system)
  - `PATH=/opt/spec2nii/bin:$PATH; spec2nii` (prepend the executable to your path)
  - `/opt/spec2nii/bin/spec2nii` (specify the fullpath to the executable)
  - `"C:\Program Files\spec2nii\spec2nii.exe"` (use quotes to deal with whitespaces in your fullpath)

- `args`: Argument string that is passed as input to spec2nii to customize its behavior
- `anon`: Set this anonymization flag to 'y' to round off age and to discard acquisition date from the meta data
- `multiraid`: The mapVBVD argument for selecting the multiraid Twix file to load (default = 2, i.e. 2nd file)

To test the proper working of spec2nii click [Test] and see the terminal output for more helptext on its input arguments

2.7 Advanced usage

2.7.1 Customized template bidsmap

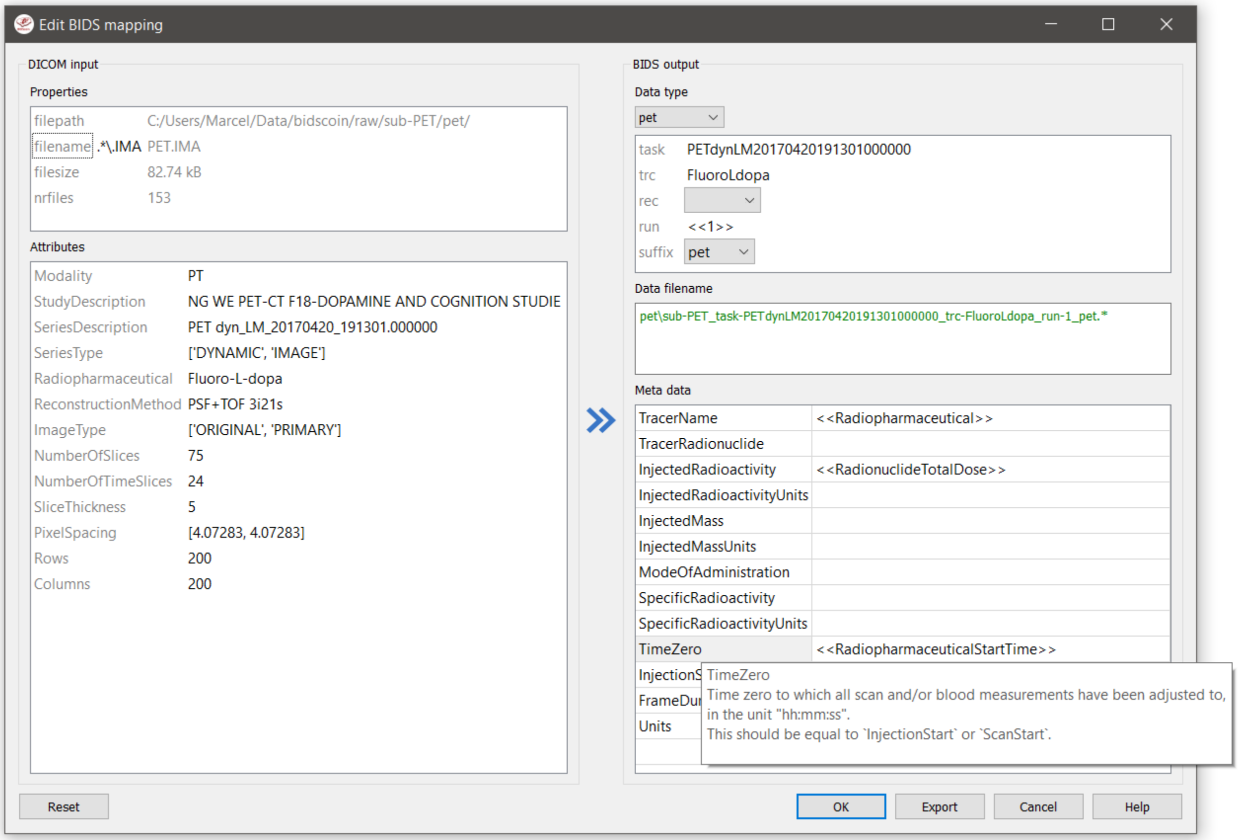
The run-items in the default 'bidsmap_dccn' template bidsmap have source dictionary values that are tailored to MRI acquisitions in the Donders Institute. Hence, if you are using different protocol parameters that do not match with these template values, then your runs will initially be data (mis)typed 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 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 to identify source data types, 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 use 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. You can still add 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 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 adviced and 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.



DICOM input

Properties

filepath		C:/Users/Marcel/Data/bidscoin/raw/sub-PET/pet/
filename	.*\.IMA	PET.IMA
filesize		82.74 kB
nrfiles		153

Attributes

Modality	PT
StudyDescription	NG WE PET-CT F18-DOPAMINE AND COGNITION STUDIE
SeriesDescription	PET dyn_LM_20170420_191301.000000
SeriesType	['DYNAMIC', 'IMAGE']
Radiopharmaceutical	Fluoro-L-dopa
ReconstructionMethod	PSF+TOF 3i21s
ImageType	['ORIGINAL', 'PRIMARY']
NumberOfSlices	75
NumberOfTimeSlices	24
SliceThickness	5
PixelSpacing	[4.07283, 4.07283]
Rows	200
Columns	200

BIDS output

Data type

pet

task	PETdynLM20170420191301000000
trc	FluoroLdopa
rec	
run	<<1>>
suffix	pet

Data filename

pet\sub-PET_task-PETdynLM20170420191301000000_trc-FluoroLdopa_run-1_pet.*

Meta data

TracerName	<<Radiopharmaceutical>>
TracerRadionuclide	
InjectedRadioactivity	<<RadionuclideTotalDose>>
InjectedRadioactivityUnits	
InjectedMass	
InjectedMassUnits	
ModeOfAdministration	
SpecificRadioactivity	
SpecificRadioactivityUnits	
TimeZero	<<RadiopharmaceuticalStartTime>>
InjectionS	
FrameDur	
Units	

TimeZero
Time zero to which all scan and/or blood measurements have been adjusted to, in the unit "hh:mm:ss".
This should be equal to `InjectionStart` or `ScanStart`.

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

&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 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
- When creating new run-items, make sure to adhere to the format defined in the BIDS schema files (`[path_to_bidscoin]/bidscoin/schema/datatypes`).

```
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
  properties: &fileattr          # This is an optional (stub) entry of filesystem␣
↪matching (could be added to any run-item)
    filepath: ~                  # File folder, e.g. ".*Parkinson.*" or ".
↪*(phantom|bottle).*"
    filename: ~                  # File name, e.g. ".*fmap.*" or ".*(fmap|field.?map|B0.?
↪map).*"
    filesize: ~                  # File size, e.g. "2[4-6]\d MB" for matching files␣
↪between 240-269 MB
    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)
```

(continues on next page)

(continued from previous page)

```yaml
    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: ~
  properties:
    <<: *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

all interactions of BIDScoin routines with source data are done via a plugin layer that abstracts away differences between source data formats. The bidsmapper and bidscoiner tools loop over the subjects/sessions in your source data repository and then use the plugins listed in the bidsmap to do the actual work. A few plugins come pre-installed:

- **Dcm2niix2bids: a plugin for DICOM and PAR/XML data**. The 'dcm2niix2bids' plugin is a wrapper around the well-known pydicom, nibabel and dcm2niix tools to interact with and convert DICOM and Philips PAR(/REC)/XML source data. Pydicom is used to read DICOM attributes, nibabel is used to read PAR/XML attribute values and dcm2niix is used to convert the DICOM and PAR/XML source data to NIfTI and create BIDS sidecar files. These sidecar files contain standard metadata but, to give more control to the user, this metadata is appended or overwritten by the data in the BIDS-mapping meta dictionary.
- **Spec2nii2bids: a plugin for MR spectroscopy data**. The 'spec2nii2bids' plugin is a wrapper around the recent spec2nii Python library to interact with and convert MR spectroscopy source data. Presently, the spec2nii2bids plugin is a first implementation that supports the conversion to BIDS for Philips SPAR/SDAT files, Siemens Twix files and GE P-files. As with the dcm2niix2bids plugin, the produced sidecar files already contain standard metadata that is complemented or overruled by the meta data that users specified in the bidseditor.
- **Phys2bidscoin: a plugin for physiological data**. The 'phys2bidscoin' plugin is a wrapper around the phys2bids Python library to interact with and convert physiological source data. Phys2bids currently supports the conversion of labchart (ADInstruments) and AcqKnowledge (BIOPAC) source files to compressed tab-separated value (`.tsv.gz`) files and create their json sidecar files, as per BIDS specifications. As in the other plugins, the sidecar files contain standard metadata that is overwritten by the user data entered in the bidseditor. This plugin has been

developed during the OHBM hackathon 2021 and is still considered experimental.

Plugin programming interface

This paragraph describes the requirements and structure of plugins in order to allow advanced users and developers to write their own plugin and extent or customize BIDScoin to their needs. As can be seen in the API code snippet below (but aso see the default plugins for reference implementation), a BIDScoin plugin is a Python module with the following programming interface (functions):

Note: Run the `bidscoin` utility to list, install or uninstall BIDScoin plugins

```
"""
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 function for the plugin + its bidsmap options. Can be␣
↪called in the bidseditor
- is_sourcefile:        A function to assess whether a source file is supported by the␣
↪plugin. The return value should correspond to a data format section in the bidsmap
- get_attribute:        A function to read an attribute value from a source file
- bidsmapper_plugin:    A function to discover BIDS-mappings in a source data session.␣
↪To avoid code duplications and minimize plugin development time, various support␣
↪functions are available to the plugin programmer in BIDScoin's library module named
↪'bids'
- bidscoiner_plugin:    A function to convert a single source data session to bids␣
↪according to the specified BIDS-mappings. Various support functions are available in␣
↪the 'bids' library module
"""

import logging
from pathlib import Path

LOGGER = logging.getLogger(__name__)

# The default options that are set when installing the plugin
OPTIONS = {'command': 'demo',    # Plugin option
           'args': 'foo bar'}    # Another plugin option

# The default bids-mappings that are added when installing the plugin
BIDSMAP = {'DemoFormat':{
    'subject': '<<filepath:/sub-(.*?)/>>',     # This filesystem property extracts the␣
↪subject label from the source directory. NB: Any property or attribute can be used as␣
↪subject-label, e.g. <PatientID>
    'session': '<<filepath:/ses-(.*?)/>>',     # This filesystem property extracts the␣
↪session label from the source directory. NB: Any property or attribute can be used as␣
↪session-label, e.g. <StudyID>
```

(continues on next page)

(continued from previous page)

```
    'func': [                   # ----------------------- All functional runs -----------
↪---------
        {'provenance': '',      # The fullpath name of the source file from which the␣
↪attributes and properties are read. Serves also as a look-up key to find a run in the␣
↪bidsmap
         'properties':          # The matching (regexp) criteria go in here
            {'filepath': '',    # File folder, e.g. ".*Parkinson.*" or ".
↪*(phantom|bottle).*"
             'filename': '',    # File name, e.g. ".*fmap.*" or ".*(fmap|field.?map|B0.?
↪map).*"
             'filesize': '',    # File size, e.g. "2[4-6]\d MB" for matching files␣
↪between 240-269 MB
             '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':          # The matching (regexp) criteria go in here
            {'ch_num': '.*',
             'filetype': '.*',
             'freq': '.*',
             'ch_name': '.*',
             'units': '.*',
             'trigger_idx': '.*'},
         'bids':
            {'task': '',
             'acq': '',
             'ce': '',
             'dir': '',
             'rec': '',
             'run': '<<1>>',    # This will be updated during bidscoiner runtime (as it␣
↪depends on the already existing files)
             'recording': '',
             'suffix': 'physio'},
         'meta':                # This is an optional entry for meta-data dictionary␣
↪that are appended to the json sidecar files
            {'TriggerChannel': '<<trigger_idx>>',
             'ExpectedTimepoints': '<<num_timepoints_found>>',
             'ChannelNames': '<<ch_name>>',
             'Threshold': '<<thr>>',
             'TimeOffset': '<<time_offset>>'}}],

    [...]

    'exclude': [  # ----------------------- Data that will be left out -------------
        {'provenance': '',
         'properties':
            {'filepath': '',
             'filename': '',
             'filesize': '',
             'nrfiles': ''},
         'attributes':
            {'ch_num': '.*',
             'filetype': '.*',
```

(continues on next page)

(continued from previous page)

```
             'freq': '.*',
             'ch_name': '.*',
             'units': '.*',
             'trigger_idx': '.*'},
         'bids':
            {'task': '',
             'acq': '',
             'ce': '',
             'dir': '',
             'rec': '',
             'run': '<<1>>',
             'recording': '',
             'suffix': 'physio'},
         'meta':
            {'TriggerChannel': '<<trigger_idx>>',
             'ExpectedTimepoints': '<<num_timepoints_found>>',
             'ChannelNames': '<<ch_name>>',
             'Threshold': '<<thr>>',
             'TimeOffset': '<<time_offset>>'}}]}}


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
    :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 ''
```

(continues on next page)

(continued from previous page)

```
def get_attribute(dataformat: str, sourcefile: Path, attribute: str, options: dict) ->␣
↪Union[str, int]:
    """
    This plugin function reads attributes from the supported sourcefile

    :param dataformat:  The bidsmap-dataformat of the sourcefile, e.g. DICOM of PAR
    :param sourcefile:  The sourcefile from which the attribute value should be read
    :param attribute:   The attribute key for which the value should be read
    :param options:     A dictionary with the plugin options, e.g. taken from the␣
↪bidsmap['Options']
    :return:            The 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 dcm2niix2bids plugin for reference implementation

    :param session:     The full-path name of the subject/session raw data source folder
    :param bidsmap_new: The new study bidsmap that we are building
    :param bidsmap_old: The previous study bidsmap that has precedence over the template␣
↪bidsmap
    :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, bidsses: Path) -> None:
    """
    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:
```

(continues on next page)

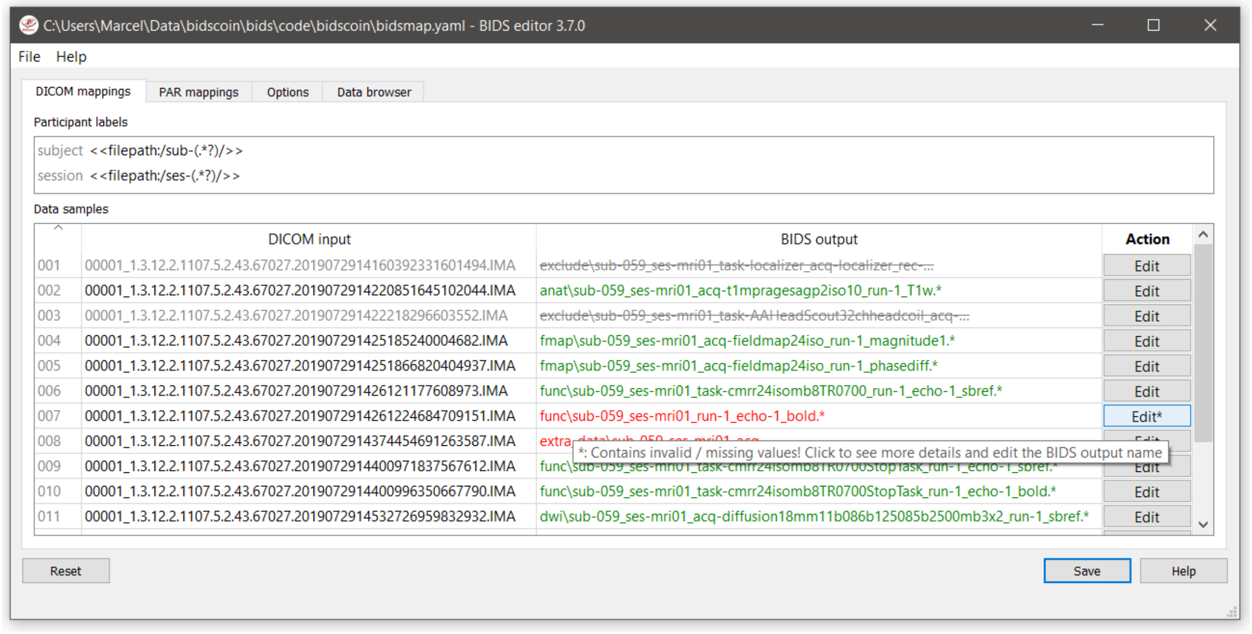
(continued from previous page)

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

    See also the dcm2niix2bids plugin for reference implementation

    :param session:     The full-path name of the subject/session source folder
    :param bidsmap:     The full mapping heuristics from the bidsmap YAML-file
    :param bidsses:     The full-path name of the BIDS output `ses-` folder
    :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


C:\Users\Marcel\Data\bidscoin\bids\code\bidscoin\bidsmap.yaml - BIDS editor 3.7.0
File
Help
DICOM mappings
PAR mappings
Options
Data browser
Reset
Save
Help

Participant labels

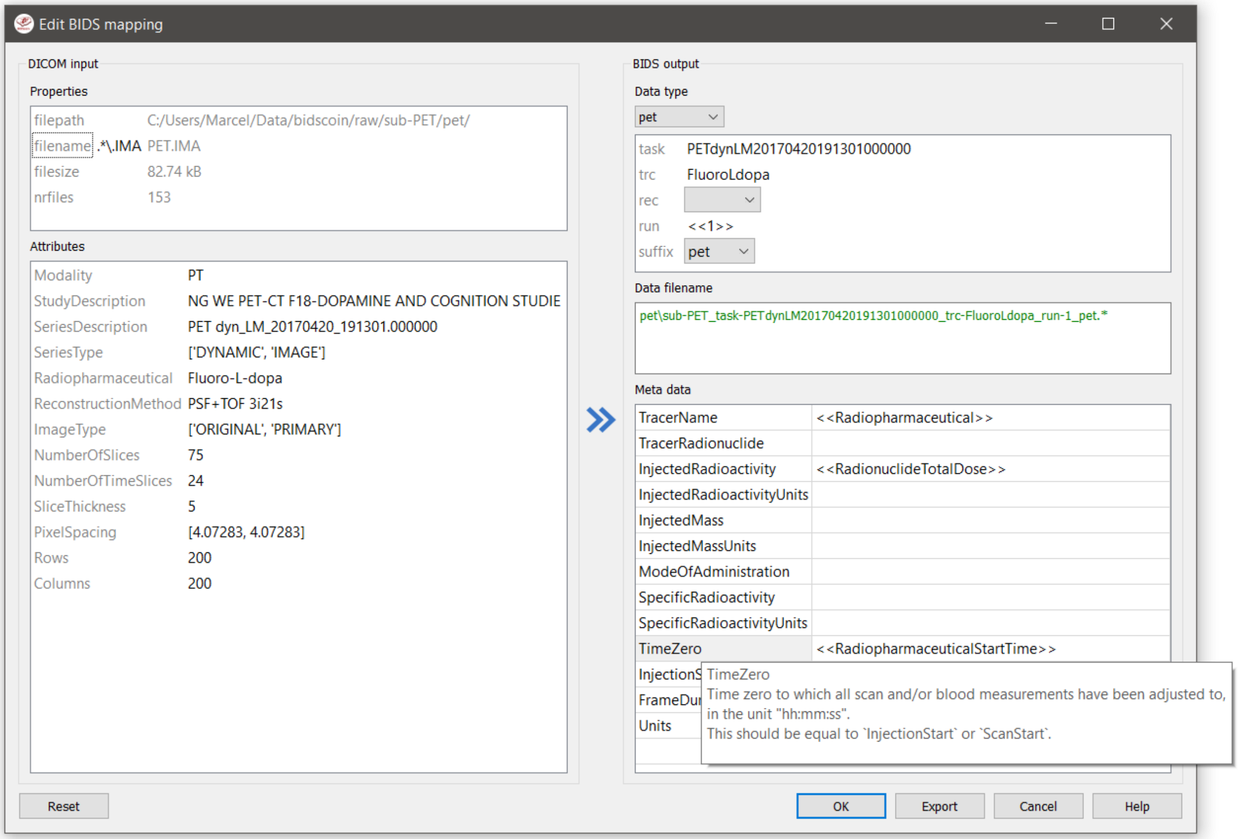
subject <<filepath:/sub-(.*?)/>>
session <<filepath:/ses-(.*?)/>>

Data samples

	DICOM input	BIDS output	Action
001	00001_1.3.12.2.1107.5.2.43.67027.2019072914160392331601494.IMA	~~exclude\sub-059_ses-mri01_task-localizer_acq-localizer_rec-...~~	Edit
002	00001_1.3.12.2.1107.5.2.43.67027.2019072914220851645102044.IMA	anat\sub-059_ses-mri01_acq-t1mpragesagp2iso10_run-1_T1w.*	Edit
003	00001_1.3.12.2.1107.5.2.43.67027.2019072914222182966035552.IMA	~~exclude\sub-059_ses-mri01_task-AAHeadScout32chheadcoil_acq-...~~	Edit
004	00001_1.3.12.2.1107.5.2.43.67027.201907291425185240004682.IMA	fmap\sub-059_ses-mri01_acq-fieldmap24iso_run-1_magnitude1.*	Edit
005	00001_1.3.12.2.1107.5.2.43.67027.2019072914251866820404937.IMA	fmap\sub-059_ses-mri01_acq-fieldmap24iso_run-1_phasediff.*	Edit
006	00001_1.3.12.2.1107.5.2.43.67027.201907291426121177608973.IMA	func\sub-059_ses-mri01_task-cmrr24isomb8TR0700_run-1_echo-1_sbref.*	Edit
007	00001_1.3.12.2.1107.5.2.43.67027.2019072914261224684709151.IMA	func\sub-059_ses-mri01_run-1_echo-1_bold.*	Edit*
008	00001_1.3.12.2.1107.5.2.43.67027.2019072914374454691263587.IMA	extra_data\sub-059_ses-mri01_acq	Edit
009	00001_1.3.12.2.1107.5.2.43.67027.2019072914400971837567612.IMA	func\sub-059_ses-mri01_task-cmrr24isomb8TR0700StopTask_run-1_echo-1_sbref.*	Edit
010	00001_1.3.12.2.1107.5.2.43.67027.2019072914400996350667790.IMA	func\sub-059_ses-mri01_task-cmrr24isomb8TR0700StopTask_run-1_echo-1_bold.*	Edit
011	00001_1.3.12.2.1107.5.2.43.67027.2019072914532726959832932.IMA	dwi\sub-059_ses-mri01_acq-diffusion18mm11b086b125085b2500mb3x2_run-1_sbref.*	Edit

*: Contains invalid / missing values! Click to see more details and edit the BIDS output name

Fig. 8: The bidseditor main window with an overview of the data types in the source data (left column) with a preview of the BIDS output names (right column). The green or red color indicates whether manual editing of the BIDS-mapping is necessary, while the strikeout text indicates that the datatype will not be converted, which is useful for handling irrelevant data. The user can edit the `subject` and `session` property values if needed (`session` can be left empty to be omitted) and the result is immediately reflected in the preview. Different tabs represent different data formats in the source dataset, i.e. DICOM and PAR, which are represented as separate sections in the bidsmap. In addition, there is a tab to edit the study specific options and a tab in which the user can browse the organization of the source data and inspect the data.



Properties

filepath		C:/Users/Marcel/Data/bidscoin/raw/sub-PET/pet/
filename	.*\.IMA	PET.IMA
filesize		82.74 kB
nrfiles		153

Attributes

Modality	PT
StudyDescription	NG WE PET-CT F18-DOPAMINE AND COGNITION STUDIE
SeriesDescription	PET dyn_LM_20170420_191301.000000
SeriesType	['DYNAMIC', 'IMAGE']
Radiopharmaceutical	Fluoro-L-dopa
ReconstructionMethod	PSF+TOF 3i21s
ImageType	['ORIGINAL', 'PRIMARY']
NumberOfSlices	75
NumberOfTimeSlices	24
SliceThickness	5
PixelSpacing	[4.07283, 4.07283]
Rows	200
Columns	200

Data type: pet

task	PETdynLM20170420191301000000
trc	FluoroLdopa
rec	
run	<<1>>
suffix	pet

Data filename

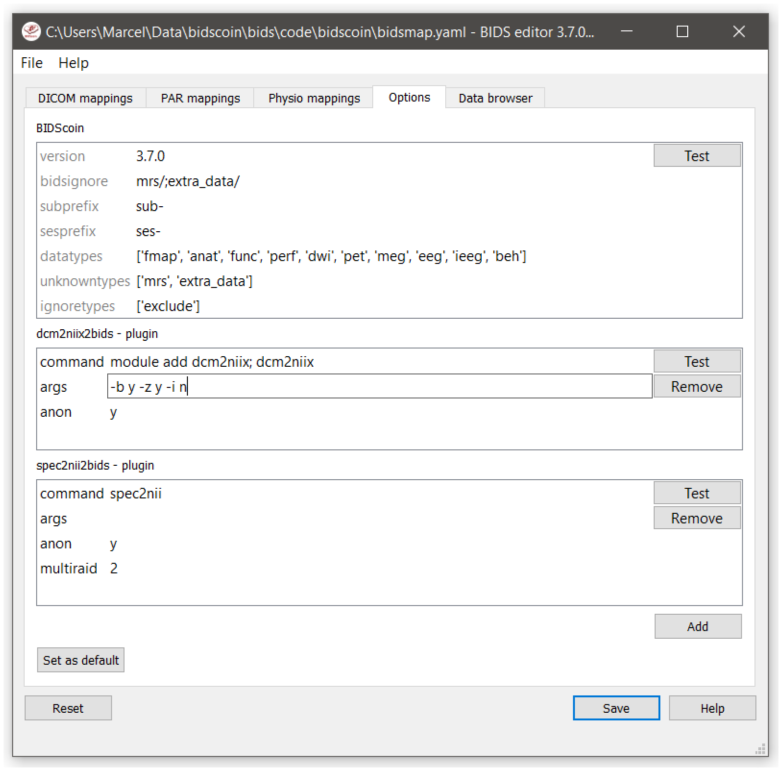
pet\sub-PET_task-PETdynLM20170420191301000000_trc-FluoroLdopa_run-1_pet.*

Meta data

TracerName	<<Radiopharmaceutical>>
TracerRadionuclide	
InjectedRadioactivity	<<RadionuclideTotalDose>>
InjectedRadioactivityUnits	
InjectedMass	
InjectedMassUnits	
ModeOfAdministration	
SpecificRadioactivity	
SpecificRadioactivityUnits	
TimeZero	<<RadiopharmaceuticalStartTime>>
InjectionS	
FrameDur	
Units	

TimeZero
Time zero to which all scan and/or blood measurements have been adjusted to, in the unit "hh:mm:ss".
This should be equal to `InjectionStart` or `ScanStart`.

Fig. 9: The BIDS-mapping edit window featuring file name matching (`.*\.IMA`) and dynamic metadata values (e.g. `TimeZero`). BIDS values that are restricted to a limited set are presented with a drop-down menu (here labeled [Data type], [rec] and [suffix]). The user can immediately see the results of their edits in the preview of the BIDS output filename. A green filename indicates that the name is compliant with the BIDS standard, whereas a red name indicates that the user still needs to fill out one or more compulsory bids values (with a pop-up window appearing if the user ignores it). Hoovering with the mouse over features explanatory text from the BIDS schema files. Double clicking on the DICOM filename opens a new window displaying the full header information with all attributes. The user can export the customized mapping to a different bidsmap on disk.



BIDScoin

version	3.7.0
bidsignore	mrs/;extra_data/
subprefix	sub-
sesprefix	ses-
datatypes	['fmap', 'anat', 'func', 'perf', 'dwi', 'pet', 'meg', 'eeg', 'ieeg', 'beh']
unknowntypes	['mrs', 'extra_data']
ignoretypes	['exclude']

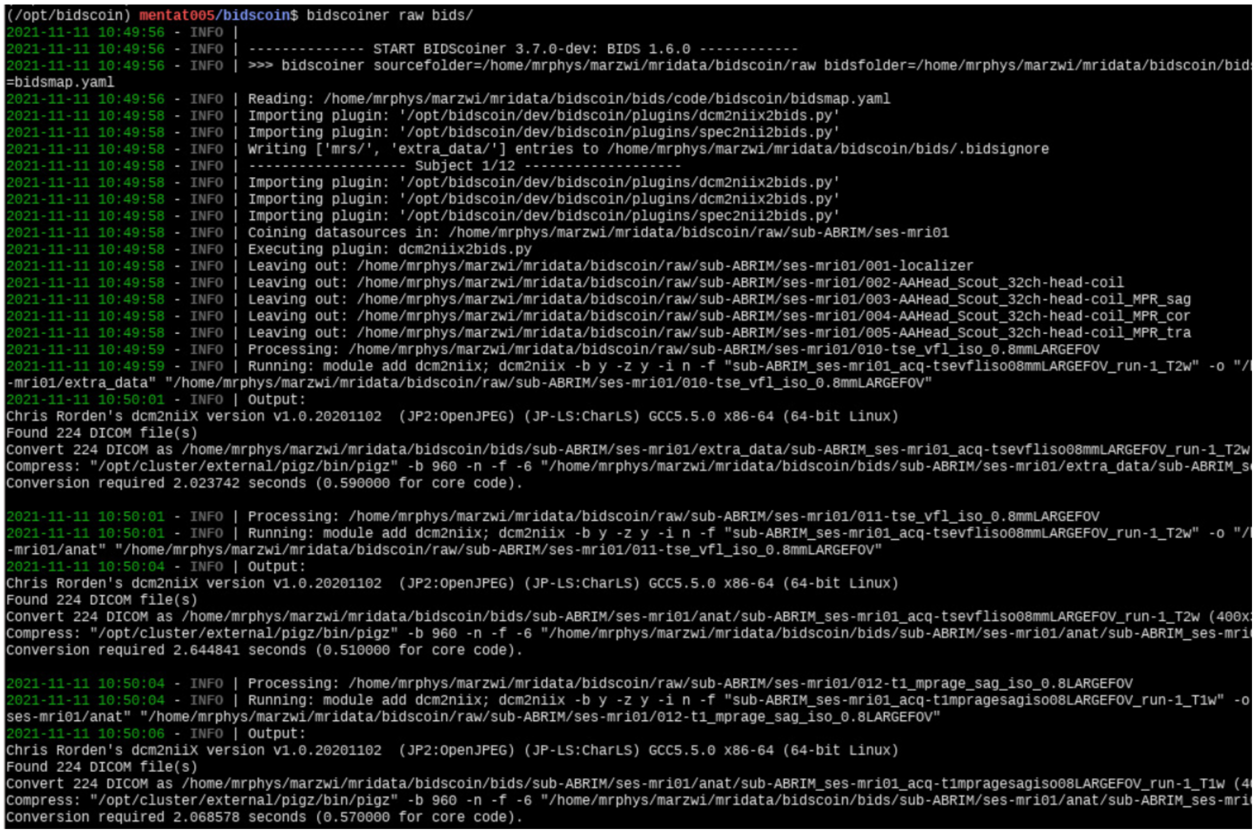
dcm2niix2bids - plugin

command	module add dcm2niix; dcm2niix
args	-b y -z y -i n
anon	y

spec2nii2bids - plugin

command	spec2nii
args	
anon	y
multiraid	2

Fig. 10: The bidsmap options for BIDScoin and its plugins. Note how the GUI automatically adapts with a new [Physio mappings] tab due to the presence of physiological data in the repository, i.e. to the presence of a ‘Physio’ section in the study bidsmap. The user can manage the plugins that will be used with the [Add] and [Remove] buttons, and save the current options to the template bidsmap by using the [Set as default] button.



```
(/opt/bidscoin) mentat005/bidscoin$ bidscoiner raw bids/
2021-11-11 10:49:56 - INFO |
2021-11-11 10:49:56 - INFO | -------------- START BIDScoiner 3.7.0-dev: BIDS 1.6.0 ------------
2021-11-11 10:49:56 - INFO | >>> bidscoiner sourcefolder=/home/mrphys/marzwi/mridata/bidscoin/raw bidsfolder=/home/mrphys/marzwi/mridata/bidscoin/bid
=bidsmap.yaml
2021-11-11 10:49:56 - INFO | Reading: /home/mrphys/marzwi/mridata/bidscoin/bids/code/bidscoin/bidsmap.yaml
2021-11-11 10:49:58 - INFO | Importing plugin: '/opt/bidscoin/dev/bidscoin/plugins/dcm2niix2bids.py'
2021-11-11 10:49:58 - INFO | Importing plugin: '/opt/bidscoin/dev/bidscoin/plugins/spec2nii2bids.py'
2021-11-11 10:49:58 - INFO | Writing ['mrs/', 'extra_data/'] entries to /home/mrphys/marzwi/mridata/bidscoin/bids/.bidsignore
2021-11-11 10:49:58 - INFO | ------------------- Subject 1/12 -------------------
2021-11-11 10:49:58 - INFO | Importing plugin: '/opt/bidscoin/dev/bidscoin/plugins/dcm2niix2bids.py'
2021-11-11 10:49:58 - INFO | Importing plugin: '/opt/bidscoin/dev/bidscoin/plugins/dcm2niix2bids.py'
2021-11-11 10:49:58 - INFO | Importing plugin: '/opt/bidscoin/dev/bidscoin/plugins/spec2nii2bids.py'
2021-11-11 10:49:58 - INFO | Coining datasources in: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01
2021-11-11 10:49:58 - INFO | Executing plugin: dcm2niix2bids.py
2021-11-11 10:49:58 - INFO | Leaving out: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/001-localizer
2021-11-11 10:49:58 - INFO | Leaving out: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/002-AAHead_Scout_32ch-head-coil
2021-11-11 10:49:58 - INFO | Leaving out: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/003-AAHead_Scout_32ch-head-coil_MPR_sag
2021-11-11 10:49:58 - INFO | Leaving out: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/004-AAHead_Scout_32ch-head-coil_MPR_cor
2021-11-11 10:49:58 - INFO | Leaving out: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/005-AAHead_Scout_32ch-head-coil_MPR_tra
2021-11-11 10:49:59 - INFO | Processing: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/010-tse_vfl_iso_0.8mmLARGEFOV
2021-11-11 10:49:59 - INFO | Running: module add dcm2niix; dcm2niix -b y -z y -i n -f "sub-ABRIM_ses-mri01_acq-tsevfliso08mmLARGEFOV_run-1_T2w" -o "/
-mri01/extra_data" "/home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/010-tse_vfl_iso_0.8mmLARGEFOV"
2021-11-11 10:50:01 - INFO | Output:
Chris Rorden's dcm2niiX version v1.0.20201102  (JP2:OpenJPEG) (JP-LS:CharLS) GCC5.5.0 x86-64 (64-bit Linux)
Found 224 DICOM file(s)
Convert 224 DICOM as /home/mrphys/marzwi/mridata/bidscoin/bids/sub-ABRIM/ses-mri01/extra_data/sub-ABRIM_ses-mri01_acq-tsevfliso08mmLARGEFOV_run-1_T2w
Compress: "/opt/cluster/external/pigz/bin/pigz" -b 960 -n -f -6 "/home/mrphys/marzwi/mridata/bidscoin/bids/sub-ABRIM/ses-mri01/extra_data/sub-ABRIM_s
Conversion required 2.023742 seconds (0.590000 for core code).

2021-11-11 10:50:01 - INFO | Processing: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/011-tse_vfl_iso_0.8mmLARGEFOV
2021-11-11 10:50:01 - INFO | Running: module add dcm2niix; dcm2niix -b y -z y -i n -f "sub-ABRIM_ses-mri01_acq-tsevfliso08mmLARGEFOV_run-1_T2w" -o "/
-mri01/anat" "/home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/011-tse_vfl_iso_0.8mmLARGEFOV"
2021-11-11 10:50:04 - INFO | Output:
Chris Rorden's dcm2niiX version v1.0.20201102  (JP2:OpenJPEG) (JP-LS:CharLS) GCC5.5.0 x86-64 (64-bit Linux)
Found 224 DICOM file(s)
Convert 224 DICOM as /home/mrphys/marzwi/mridata/bidscoin/bids/sub-ABRIM/ses-mri01/anat/sub-ABRIM_ses-mri01_acq-tsevfliso08mmLARGEFOV_run-1_T2w (400x
Compress: "/opt/cluster/external/pigz/bin/pigz" -b 960 -n -f -6 "/home/mrphys/marzwi/mridata/bidscoin/bids/sub-ABRIM/ses-mri01/anat/sub-ABRIM_ses-mri
Conversion required 2.644841 seconds (0.510000 for core code).

2021-11-11 10:50:04 - INFO | Processing: /home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/012-t1_mprage_sag_iso_0.8LARGEFOV
2021-11-11 10:50:04 - INFO | Running: module add dcm2niix; dcm2niix -b y -z y -i n -f "sub-ABRIM_ses-mri01_acq-t1mpragesagiso08LARGEFOV_run-1_T1w" -o
ses-mri01/anat" "/home/mrphys/marzwi/mridata/bidscoin/raw/sub-ABRIM/ses-mri01/012-t1_mprage_sag_iso_0.8LARGEFOV"
2021-11-11 10:50:06 - INFO | Output:
Chris Rorden's dcm2niiX version v1.0.20201102  (JP2:OpenJPEG) (JP-LS:CharLS) GCC5.5.0 x86-64 (64-bit Linux)
Found 224 DICOM file(s)
Convert 224 DICOM as /home/mrphys/marzwi/mridata/bidscoin/bids/sub-ABRIM/ses-mri01/anat/sub-ABRIM_ses-mri01_acq-t1mpragesagiso08LARGEFOV_run-1_T1w (4
Compress: "/opt/cluster/external/pigz/bin/pigz" -b 960 -n -f -6 "/home/mrphys/marzwi/mridata/bidscoin/bids/sub-ABRIM/ses-mri01/anat/sub-ABRIM_ses-mri
Conversion required 2.068578 seconds (0.570000 for core code).
```

Fig. 11: Snapshot of running *bidscoiner* in the terminal.

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. Getting started

Depending on how BIDScoin was installed, you may have to set your Python environment settings before you can run BIDScoin commands from your command-line interface / shell. In the DCCN compute cluster example below it is assumed that an environment module is used to load your Linux Anaconda Python installation and that BIDScoin is installed in a conda environment named "bidscoin". Run or adjust these commands to your computer system if needed:

```
$ 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
```

Now you should be able to execute BIDScoin commands. Test this by running `bidscoin` to get a general overview. Can you generate a list of all BIDScoin tools? What about the plugins?

2. Data preparation

Create a tutorial playground folder by executing these shell commands:

```
$ bidscoin --download .            # Download the tutorial data (use a "." for the␣
↪current folder or a pathname of choice to save it elsewhere)
$ cd ./bidscointutorial            # Go to the downloaded data (replace "." with the␣
↪full pathname if your data was saved elsewhere)
```

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
```

(continues on next page)

(continued from previous page)

```
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

3. 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 at the end, 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 acquisition 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.

4. 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`.

5. 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 deface 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 3.7.1-dev

2.10.2 3.7.0 - 2021-12-20

Added

- A BIDScoin installation test (`bidscoin -t`)
- Option to install extra packages, such as phys2bids
- A bidseditor button to save the Options to a (default) template bidsmap
- Sub-/ses-prefix settings and BIDS / extra_data / excluded datatypes in bidsmap['Options']['bidscoin']
- Regular expressions for extracting property and attribute substrings from dynamic values via a <key:regular_expression> syntax
- A plugin for spec2nii to convert MR spectroscopy data
- An experimental plugin for phys2bids to convert physiological data
- An experimental plugin for pet2bids to convert MR spectroscopy data
- Added a multi-echo deface function `medeface` that uses the same defacemask for all echo-images
- The possibility to extract DICOM values using pydicom-style tag numbers (in addition to the attribute name)
- The possibility for plugins to set default bidsmappings and Options when installed
- A Singularity container configuration file
- Improved (more fine-grained) plugin installation procedures

- The option to remove decimals from age and discard acquisition dates from the meta data

Changed

- Plugins should now have a `is_sourcefile` and a `get_attribute` function and have a simpler/changed API (-> DataSource class)
- The intricate filtering of the `nrfiles` property by the other filesystem properties has been removed and is now a pure/unfiltered file-system property
- The default <<`SourceFilePath`>> keyword has been replaced by the more flexible <filepath:/sub-(.*?)/> property to extract the subject / session label
- The dcm2bidsmap and the dcm2niix2bids plugins have been merged
- The dicomsort utility has new naming-scheme functionality
- Removed the obsolete bidsmap_template.yaml file

Fixed

- Avoid storing Python literal structures as strings

2.10.3 3.6.3 - 2021-06-14

Fixed

Remove regular expression metacharacters from the source attribute if needed (could cause a regexp compile error) Fixed for list of dynamic <> fields

2.10.4 3.6.2 - 2021-05-31

Fixed

Removed the redundant importlib dependency from the requirements (could cause an installation error)

2.10.5 3.6.1 - 2021-05-20

Fixed

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

2.10.6 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`)

2.10.7 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

2.10.8 3.5.2 - 2021-03-21

Fixed:

- pypi upload

2.10.9 3.5.1 - 2021-03-21

Added

- BIDScoin version update checks

Fixed

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

2.10.10 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

2.10.11 3.0.8 - 2020-09-28

Fixed

- Various minor bugs

2.10.12 3.0.6 - 2020-08-05

Fixed

- Minor but important bugfix in the setup :-)

2.10.13 3.0.5 - 2020-08-05

Added

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

Fixed

- Various bugs

2.10.14 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

2.10.15 3.0.3 - 2020-04-14

Fixed

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

2.10.16 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 :-)

2.10.17 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

2.10.18 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)

2.10.19 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 :-))

2.10.20 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

2.10.21 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

2.10.22 2.1 - 2019-06-23

Added

- Editing of bidsmap Options

Fixed

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

2.10.23 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

2.10.24 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

2.10.25 1.4 - 2018-10-22

Added

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

2.10.26 1.3 - 2018-09-28

Changed

- Refactored bidsmap naming

Fixed

- Various bugs

2.10.27 1.2 - 2018-09-14

Added

- Improved fieldmap support

Changed

- Yaml-syntax

2.10.28 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