heudiconv Documentation

Release 0.13.0

Heudiconv team

May 08, 2023

Contents

a heuristic-centric DICOM converter

CHAPTER 1

About

`heudiconv` is a flexible DICOM converter for organizing brain imaging data into structured directory layouts.

- It allows flexible directory layouts and naming schemes through customizable heuristics implementations.
- It only converts the necessary DICOMs and ignores everything else in a directory.
- You can keep links to DICOM files in the participant layout.
- Using dcm2niix under the hood, it's fast.
- It can track the provenance of the conversion from DICOM to NIfTI in W3C PROV format.
- It provides assistance in converting to BIDS.
- It integrates with DataLad to place converted and original data under git/git-annex version control while automatically annotating files with sensitive information (e.g., non-defaced anatomicals, etc).

CHAPTER 2

Installation

See our installation page on heudiconv.readthedocs.io .

CHAPTER 3

HOWTO 101

In a nutshell – `heudiconv` operates using a heuristic which, given metadata from DICOMs, would decide how to name resultant (from conversion using dcm2niix) files. Heuristic convertall could actually be used with no real heuristic and by simply establish your own conversion mapping through editing produced mapping files. In most use-cases of retrospective study data conversion, you would need to create your custom heuristic following existing heuristics as examples and/or referring to "Heuristic" section in the documentation. **Note** that ReproIn heuristic is generic and powerful enough to be adopted virtually for *any* study: For prospective studies, you would just need to name your sequences following the ReproIn convention, and for retrospective conversions, you often would be able to create a new versatile heuristic by simply providing remappings into ReproIn as shown in this issue (documentation is coming).

Having decided on a heuristic, you could use the command line:

```
heudiconv -f HEURISTIC-FILE-OR-NAME -o OUTPUT-PATH --files INPUT-PATHs
```

with various additional options (see `heudiconv --help` or "Usage" in documentation) to tune its behavior to convert your data.

For detailed examples and guides, please check out ReproIn conversion invocation examples and the user tutorials in the documentation.

CHAPTER 4

How to cite

Please use Zenodo record for your specific version of HeuDiConv. We also support gathering all relevant citations via DueCredit.

CHAPTER 5

How to contribute

For a detailed into, see our contributing guide.

Our releases are packaged using Intuit auto, with the corresponding workflow including Docker image preparation being found in `.github/workflows/release.yml`.

CHAPTER 6

Contributing to HeuDiConv

6.1 Files organization

- heudiconv/ is the main Python module where major development is happening, with major submodules being:
  - `cli/` - wrappers and argument parsers bringing the HeuDiConv functionality to the command line.
  - `external/` - general compatibility layers for external functions HeuDiConv depends on.
  - `heuristics/` - heuristic evaluators for workflows, pull requests here are particularly welcome.
- docs/ - documentation directory.
- utils/ - helper utilities used during development, testing, and distribution of HeuDiConv.

6.2 How to contribute

The preferred way to contribute to the HeuDiConv code base is to fork the main repository on GitHub.

If you are unsure what that means, here is a set-up workflow you may wish to follow:

0. Fork the project repository on GitHub, by clicking on the "Fork" button near the top of the page — this will create a copy of the repository writeable by your GitHub user.
1. Set up a clone of the repository on your local machine and connect it to both the "official" and your copy of the repository on GitHub:

```
git clone git://github.com/nipy/heudiconv
cd heudiconv
git remote rename origin official
git remote add origin git://github.com/YOUR_GITHUB_USERNAME/heudiconv
```

2. When you wish to start a new contribution, create a new branch:

```
git checkout -b topic_of_your_contribution
```

3. When you are done making the changes you wish to contribute, record them in Git:

```
git add the/paths/to/files/you/modified can/be/more/than/one
git commit
```

3. Push the changes to your copy of the code on GitHub, following which Git will provide you with a link which you can click to initiate a pull request:

```
git push -u origin topic_of_your_contribution
```

(If any of the above seems overwhelming, you can look up the Git documentation on the web.)

6.3 Development environment

We support Python 3 only (>= 3.7).

Dependencies which you will need are listed in the repository. Note that you will likely have these will already be available on your system if you used a package manager (e.g. Debian's `apt-get`, Gentoo's `emerge`, or simply PIP) to install the software.

Development work might require live access to the copy of HeuDiConv which is being developed. If a system-wide release of HeuDiConv is already installed, or likely to be, it is best to keep development work sandboxed inside a dedicated virtual environment. This is best accomplished via:

```
cd /path/to/your/clone/of/heudiconv
mkdir -p venvs/dev
python -m venv venvs/dev
source venvs/dev/bin/activate
pip install -e .[all]
```

6.4 Additional Hints

It is recommended to check that your contribution complies with the following rules before submitting a pull request:

- All public functions (i.e. functions whose name does not start with an underscore) should have informative docstrings with sample usage presented as doctests when appropriate.
- Docstrings are formatted in NumPy style.
- Lines are no longer than 120 characters.
- All tests still pass:

```
cd /path/to/your/clone/of/heudiconv
pytest -vvs .
```

- New code should be accompanied by new tests.

6.5 Contents

6.5.1 Installation

`Heudiconv` is packaged and available from many different sources.

Local

Released versions of HeuDiConv are available on PyPI and conda. If installing through `PyPI`, eg:

```
pip install heudiconv[all]
```

Manual installation of dcm2niix is required. You can also benefit from an installer/downloader helper `dcm2niix` package on `PyPI`, so you can simply `pip install dcm2niix` if you are installing in user space so subsequently it would be able to download and install dcm2niix binary.

On Debian-based systems, we recommend using NeuroDebian, which provides the heudiconv package.

Docker

If Docker is available on your system, you can visit our page on Docker Hub to view available releases. To pull the latest release, run:

```
$ docker pull nipy/heudiconv:latest
```

Note that when using HeuDiConv via `docker run`, you might need to provide your user and group IDs so they map correspondingly within the container, i.e.:

```
$ docker run --user=$(id -u):$(id -g) -e "UID=$(id -u)" -e "GID=$(id -g)" --rm -t -v
↪$PWD:$PWD nipy/heudiconv:latest [OPTIONS TO FOLLOW]
```

Additionally, HeuDiConv is available through the Docker image at repronim/reproin provided by ReproIn heuristic project, which develops the `reproin` heuristic.

Singularity

If Singularity is available on your system, you can use it to pull and convert our Docker images! For example, to pull and build the latest release, you can run:

```
$ singularity pull docker://nipy/heudiconv:latest
```

Singularity YODA style using ///repronim/containers

ReproNim provides a large collection of Singularity container images of popular neuroimaging tools, e.g. all the BIDS-Apps. This collection also includes the forementioned container images for HeuDiConv and ReproIn in the Singularity image format. This collection is available as a DataLad dataset at ///repronim/containers on datasets.datalad.org and as a GitHub repo. The HeuDiConv and ReproIn container images are named `nipy-heudiconv` and `repronim-reproin`, respectively, in this collection. To use them, you can install the DataLad dataset and then use the `datalad containers-run` command to run. For a more detailed example of using images from this collection while fulfilling the YODA Principles, please check out A typical YODA workflow in the documentation of this collection.

Note: With the `datalad containers-run` command, the images in this collection work on macOS (OSX) as well for `reproniм/containers` helpers automagically take care of running the Singularity containers via Docker.

6.5.2 Changes

```
# v0.13.0 (Mon May 08 2023)

####  Enhancement

- Add type annotations [#656](https://github.com/nipy/heudiconv/pull/656)␣
↪([@jwodder](https://github.com/jwodder) [@yarikoptic](https://github.com/
↪yarikoptic))
- ENH: Support extracting DICOMs from ZIP files (and possibly other archives) by␣
↪switching to use shutil.unpack_archive instead of tarfile module functionality [
↪#471](https://github.com/nipy/heudiconv/pull/471) ([@HippocampusGirl](https://
↪github.com/HippocampusGirl) [@psadil](https://github.com/psadil))
- Allow filling of acq_time when AcquisitionDate AcquisitionTime missing [
↪#614](https://github.com/nipy/heudiconv/pull/614) ([@psadil](https://github.com/
↪psadil))

####  Bug Fix

- BF(?): make _setter images be taken as scouts - only DICOMs are saved [#570](https:/
↪/github.com/nipy/heudiconv/pull/570) ([@yarikoptic](https://github.com/yarikoptic))
- Adjust .mailmap to account for mapping various folks with multiple emails so that␣
↪git shortlog -sn -e provides entries without duplicates [#570](https://github.com/
↪nipy/heudiconv/pull/570) ([@yarikoptic](https://github.com/yarikoptic))
- Make an `embed_dicom_and_nifti_metadata()` annotation work on Python 3.7 [
↪#673](https://github.com/nipy/heudiconv/pull/673) ([@jwodder](https://github.com/
↪jwodder))
- Merge branch 'feature_dicom_compresslevel' [#673](https://github.com/nipy/heudiconv/
↪pull/673) ([@yarikoptic](https://github.com/yarikoptic))
- Update heudiconv/dicoms.py [#669](https://github.com/nipy/heudiconv/pull/669)␣
↪([@octomike](https://github.com/octomike))
- Don't call `logging.basicConfig()` in `__init__.py` [#659](https://github.com/nipy/
↪heudiconv/pull/659) ([@jwodder](https://github.com/jwodder))

####  Pushed to `master`

- Mailmapping more contributors ([@yarikoptic](https://github.com/yarikoptic))
- Adjust comment and remove trailing space flipping linting ([@yarikoptic](https://
↪github.com/yarikoptic))

####  Internal

- Declare `custom_grouping` return type instead of casting [#671](https://github.com/
↪nipy/heudiconv/pull/671) ([@jwodder](https://github.com/jwodder))
- Use `pydicom.dcmread()` instead of `pydicom.read_file()` [#668](https://github.com/
↪nipy/heudiconv/pull/668) ([@jwodder](https://github.com/jwodder))
- Add `sample_nifti.json` to `.gitignore` [#663](https://github.com/nipy/heudiconv/
↪pull/663) ([@jwodder](https://github.com/jwodder))
- Write command arguments as lists of strings instead of splitting strings on␣
↪whitespace [#664](https://github.com/nipy/heudiconv/pull/664) ([@jwodder](https://
↪github.com/jwodder))
- Add & apply pre-commit and lint job [#658](https://github.com/nipy/heudiconv/pull/
↪658) ([@jwodder](https://github.com/jwodder))
```

(continues on next page)

(continued from previous page)

```
- Fix some strings with \ (make them raw or double-\), improve pytest config: move to␣
↪tox.ini, make unknown warnings into errors [#660](https://github.com/nipy/heudiconv/
↪pull/660) ([@jwodder](https://github.com/jwodder))
- Replace py.path with pathlib [#654](https://github.com/nipy/heudiconv/pull/654)␣
↪([@jwodder](https://github.com/jwodder))

####  Tests

- Make `test_private_csa_header` test write to temp dir [#666](https://github.com/nipy/
↪heudiconv/pull/666) ([@jwodder](https://github.com/jwodder))

####  Dependency Updates

- Replace third-party `mock` library with stdlib's `unittest.mock` [#661](https://
↪github.com/nipy/heudiconv/pull/661) ([@jwodder](https://github.com/jwodder))
- Remove kludgy support for older versions of pydicom and dcmstack [#662](https://
↪github.com/nipy/heudiconv/pull/662) ([@jwodder](https://github.com/jwodder))
- Remove use of `six` [#655](https://github.com/nipy/heudiconv/pull/655)␣
↪([@jwodder](https://github.com/jwodder))

#### Authors: 5

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Lea Waller ([@HippocampusGirl](https://github.com/HippocampusGirl))
- Michael ([@octomike](https://github.com/octomike))
- Patrick Sadil ([@psadil](https://github.com/psadil))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.12.2 (Tue Mar 14 2023)

####  Internal

- [DATALAD RUNCMD] produce updated dockerfile [#652](https://github.com/nipy/
↪heudiconv/pull/652) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 1

- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.12.1 (Tue Mar 14 2023)

####  Bug Fix

- Re-add explicit instructions to install dcm2niix "manually" and remove it from␣
↪install_requires [#651](https://github.com/nipy/heudiconv/pull/651)␣
↪([@yarikoptic](https://github.com/yarikoptic))

####  Documentation

- Contributing guide. [#641](https://github.com/nipy/heudiconv/pull/641)␣
↪([@TheChymera](https://github.com/TheChymera))
- Reword and correct punctuation on installation.rst [#643](https://github.com/nipy/
↪heudiconv/pull/643) ([@yarikoptic](https://github.com/yarikoptic)␣
↪[@candleindark](https://github.com/candleindark))
```

(continues on next page)

(continued from previous page)

```
#### Authors: 3

- Horea Christian ([@TheChymera](https://github.com/TheChymera))
- Isaac To ([@candleindark](https://github.com/candleindark))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.12.0 (Tue Feb 21 2023)

####  Enhancement

- strip non-alphanumeric from session ids too [#647](https://github.com/nipy/
↪heudiconv/pull/647) ([@keithcallenberg](https://github.com/keithcallenberg)␣
↪[@yarikoptic](https://github.com/yarikoptic))

####  Bug Fix

- Docker images: tag also as "unstable", strip "v" prefix, and avoid building in non-
↪release workflow for releases. [#642](https://github.com/nipy/heudiconv/pull/642)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- add install link to README [#640](https://github.com/nipy/heudiconv/pull/640)␣
↪([@asmacdo](https://github.com/asmacdo))
- Setting git author and email in test environment [#631](https://github.com/nipy/
↪heudiconv/pull/631) ([@TheChymera](https://github.com/TheChymera))
- Duecredit dcm2niix [#622](https://github.com/nipy/heudiconv/pull/622)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- Do not issue warning if cannot parse _task entity [#621](https://github.com/nipy/
↪heudiconv/pull/621) ([@yarikoptic](https://github.com/yarikoptic))
- Provide codespell config and workflow [#619](https://github.com/nipy/heudiconv/pull/
↪619) ([@yarikoptic](https://github.com/yarikoptic))
- BF: Use .get in group_dicoms_into_seqinfos to not puke if SeriesDescription is␣
↪missing [#622](https://github.com/nipy/heudiconv/pull/622) ([@yarikoptic](https://
↪github.com/yarikoptic))
- DOC: do provide short version for sphinx [#609](https://github.com/nipy/heudiconv/
↪pull/609) ([@yarikoptic](https://github.com/yarikoptic))

####  Pushed to `master`

- DOC: add clarification on where docs/requirements.txt should be "installed" from␣
↪([@yarikoptic](https://github.com/yarikoptic))
- fix minor typo ([@yarikoptic](https://github.com/yarikoptic))
- DOC: fixed the comment. Original was copy/pasted from DataLad ([@yarikoptic](https:/
↪/github.com/yarikoptic))

####  Internal

- dcm2niix explicitly noted as a (PyPI) dependency and removed from being installed␣
↪via apt-get etc [#628](https://github.com/nipy/heudiconv/pull/628)␣
↪([@TheChymera](https://github.com/TheChymera) [@yarikoptic](https://github.com/
↪yarikoptic))

####  Documentation

- Reword number of intended ideas in README.rst [#639](https://github.com/nipy/
↪heudiconv/pull/639) ([@candleindark](https://github.com/candleindark))
```

(continues on next page)

(continued from previous page)

```
- Add a bash anon-cmd to be used to incrementally anonymize sids [#615](https://
↪github.com/nipy/heudiconv/pull/615) ([@yarikoptic](https://github.com/yarikoptic))
- Reword and correct punctuation on usage.rst [#644](https://github.com/nipy/
↪heudiconv/pull/644) ([@candleindark](https://github.com/candleindark)␣
↪[@yarikoptic](https://github.com/yarikoptic))
- Clarify the infotodict function [#645](https://github.com/nipy/heudiconv/pull/645)␣
↪([@yarikoptic](https://github.com/yarikoptic))
- Added distribution badges [#632](https://github.com/nipy/heudiconv/pull/632)␣
↪([@TheChymera](https://github.com/TheChymera))
- Capitalize sentences and end sentences with period [#629](https://github.com/nipy/
↪heudiconv/pull/629) ([@candleindark](https://github.com/candleindark))
- Tune up .mailmap to harmonize Pablo, Dae and Mathias [#629](https://github.com/nipy/
↪heudiconv/pull/629) ([@yarikoptic](https://github.com/yarikoptic))
- Add HOWTO 101 section, with references to ReproIn to README.rst [#623](https://
↪github.com/nipy/heudiconv/pull/623) ([@yarikoptic](https://github.com/yarikoptic))
- minor fix -- Fix use of code:: directive [#623](https://github.com/nipy/heudiconv/
↪pull/623) ([@yarikoptic](https://github.com/yarikoptic))

####  Tests

- Add 3.11 to be tested etc [#635](https://github.com/nipy/heudiconv/pull/635)␣
↪([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 5

- Austin Macdonald ([@asmacdo](https://github.com/asmacdo))
- Horea Christian ([@TheChymera](https://github.com/TheChymera))
- Isaac To ([@candleindark](https://github.com/candleindark))
- Keith Callenberg ([@keithcallenberg](https://github.com/keithcallenberg))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.6 (Thu Nov 03 2022)

####  Internal

- Delete .dockerignore [#607](https://github.com/nipy/heudiconv/pull/607)␣
↪([@jwodder](https://github.com/jwodder))

####  Documentation

- DOC: Various fixes to make RTD build the docs again [#608](https://github.com/nipy/
↪heudiconv/pull/608) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.5 (Thu Nov 03 2022)

####  Bug Fix

- Fix certificate issue as indicated in #595 [#597](https://github.com/nipy/heudiconv/
↪pull/597) ([@neurorepro](https://github.com/neurorepro))
```

(continues on next page)

(continued from previous page)

```
- BF docker build: use python3.9 (not 3.7 which gets upgraded to 3.9) and newer␣
↪dcm2niix [#596](https://github.com/nipy/heudiconv/pull/596) ([@yarikoptic](https://
↪github.com/yarikoptic))
- Fixup miniconda spec for neurodocker so it produces dockerfile now [#596](https://
↪github.com/nipy/heudiconv/pull/596) ([@yarikoptic](https://github.com/yarikoptic))

####  Internal

- Update GitHub Actions action versions [#601](https://github.com/nipy/heudiconv/pull/
↪601) ([@jwodder](https://github.com/jwodder))
- Set action step outputs via $GITHUB_OUTPUT [#600](https://github.com/nipy/heudiconv/
↪pull/600) ([@jwodder](https://github.com/jwodder))

####  Documentation

- DOC: codespell fix a few typos in code comments [#605](https://github.com/nipy/
↪heudiconv/pull/605) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 3

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Michael ([@neurorepro](https://github.com/neurorepro))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.4 (Thu Sep 29 2022)

####  Bug Fix

- install dcmstack straight from github until it is released [#593](https://github.
↪com/nipy/heudiconv/pull/593) ([@yarikoptic](https://github.com/yarikoptic))
- DOC: provide rudimentary How to contribute section in README.rst␣
↪([@yarikoptic](https://github.com/yarikoptic))

####  Pushed to `master`

- Check out a full clone when testing ([@jwodder](https://github.com/jwodder))
- Convert Travis workflow to GitHub Actions ([@jwodder](https://github.com/jwodder))
- BF(docker): replace old -tipsy with -y -all for conda clean as neurodocker does now␣
↪([@yarikoptic](https://github.com/yarikoptic))
- adjusted script for neurodocker although it does not work ([@yarikoptic](https://
↪github.com/yarikoptic))

####  Internal

- 0.9 of dcmstack was released, no need for github version [#594](https://github.com/
↪nipy/heudiconv/pull/594) ([@yarikoptic](https://github.com/yarikoptic))
- Minor face-lifts to ReproIn: align doc and code better to BIDS terms, address␣
↪deprecation warnings etc [#569](https://github.com/nipy/heudiconv/pull/569)␣
↪([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))
```

(continues on next page)

(continued from previous page)

```
---

# v0.11.3 (Thu May 12 2022)

####  Internal

- BF: add recently tests data missing from distribution [#567](https://github.com/
↪nipy/heudiconv/pull/567) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 1

- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.2 (Thu May 12 2022)

####  Internal

- Make versioningit write version to file; make setup.py read version as fallback [
↪#566](https://github.com/nipy/heudiconv/pull/566) ([@jwodder](https://github.com/
↪jwodder))
- BF: add fetch-depth: 0 to get all tags into docker builds of master [#566](https://
↪github.com/nipy/heudiconv/pull/566) ([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 2

- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.1 (Tue May 10 2022)

####  Internal

- Remove .git/ from .dockerignore so that versioning works while building docker␣
↪image [#564](https://github.com/nipy/heudiconv/pull/564) ([@yarikoptic](https://
↪github.com/yarikoptic))

#### Authors: 1

- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))

---

# v0.11.0 (Tue May 10 2022)

####  Enhancement

- RF: drop Python 3.6 (EOLed), fix dcm2niix version in neurodocker script [
↪#555](https://github.com/nipy/heudiconv/pull/555) ([@yarikoptic](https://github.com/
↪yarikoptic))
- ENH: Adds populate_intended_for for fmaps [#482](https://github.com/nipy/heudiconv/
↪pull/482) ([@pvelasco](https://github.com/pvelasco) [@yarikoptic](https://github.
↪com/yarikoptic) bids@dbic.dartmouth.edu [@neurorepro](https://github.com/
↪neurorepro))
```

(continues on next page)

(continued from previous page)

```
####  Bug Fix

- bids_ME heuristic: add test for the dataset that raised #541, add support for MEGRE␣
↪[#547](https://github.com/nipy/heudiconv/pull/547) ([@pvelasco](https://github.com/
↪pvelasco) [@yarikoptic](https://github.com/yarikoptic))
- reproin heuristic: specify POPULATE_INTENDED_FOR_OPTS [#546](https://github.com/
↪nipy/heudiconv/pull/546) ([@yarikoptic](https://github.com/yarikoptic))
- FIX: Convert sets to lists for filename updaters [#461](https://github.com/nipy/
↪heudiconv/pull/461) ([@tsalo](https://github.com/tsalo))
- Added new infofilestyle compatible with BIDS [#12](https://github.com/nipy/
↪heudiconv/pull/12) ([@chrisgorgo](https://github.com/chrisgorgo))
- try a simple fix for wrongly ordered files in tar file [#535](https://github.com/
↪nipy/heudiconv/pull/535) ([@bpinsard](https://github.com/bpinsard))
- BF: Fix the order of the 'echo' entity in the filename [#542](https://github.com/
↪nipy/heudiconv/pull/542) ([@pvelasco](https://github.com/pvelasco))
- ENH: add HeudiconvVersion to sidecar .json files [#529](https://github.com/nipy/
↪heudiconv/pull/529) ([@yarikoptic](https://github.com/yarikoptic))
- BF (TST): make anonymize_script actually output anything and map deterministically [
↪#511](https://github.com/nipy/heudiconv/pull/511) ([@yarikoptic](https://github.com/
↪yarikoptic))
- Rename DICOMCONVERT_README.md to README.md [#4](https://github.com/nipy/heudiconv/
↪pull/4) ([@satra](https://github.com/satra))

####  Pushed to `master`

- Dockerfile - use bullseye for the base and fresh dcm2niix ([@yarikoptic](https://
↪github.com/yarikoptic))

####  Internal

- Run codespell on some obvious typos [#563](https://github.com/nipy/heudiconv/pull/
↪563) ([@yarikoptic](https://github.com/yarikoptic))
- Set up auto [#558](https://github.com/nipy/heudiconv/pull/558) ([@jwodder](https://
↪github.com/jwodder) [@yarikoptic](https://github.com/yarikoptic))

####  Tests

- BF(TST): use caplog to control logging level, use python3 in shebang [#553](https://
↪github.com/nipy/heudiconv/pull/553) ([@yarikoptic](https://github.com/yarikoptic))
- BF(TST): use caplog instead of capfd for testing if we log a warning [#534](https://
↪github.com/nipy/heudiconv/pull/534) ([@yarikoptic](https://github.com/yarikoptic))
- Travis - Use bionic for the base [#533](https://github.com/nipy/heudiconv/pull/533)␣
↪([@yarikoptic](https://github.com/yarikoptic))

#### Authors: 9

- Basile ([@bpinsard](https://github.com/bpinsard))
- Chris Gorgolewski ([@chrisgorgo](https://github.com/chrisgorgo))
- DBIC BIDS Team (bids@dbic.dartmouth.edu)
- John T. Wodder II ([@jwodder](https://github.com/jwodder))
- Michael ([@neurorepro](https://github.com/neurorepro))
- Pablo Velasco ([@pvelasco](https://github.com/pvelasco))
- Satrajit Ghosh ([@satra](https://github.com/satra))
- Taylor Salo ([@tsalo](https://github.com/tsalo))
- Yaroslav Halchenko ([@yarikoptic](https://github.com/yarikoptic))
```

(continues on next page)

(continued from previous page)

```
---

# [0.10.0] - 2021-09-16

Various improvements and compatibility/support (dcm2niix, datalad) changes.

## Added

- Add "AcquisitionTime" to the seqinfo ([#487][])
- Add support for saving the Phoenix Report in the sourcedata folder ([#489][])

## Changed

- Python 3.5 EOLed, supported (tested) versions now: 3.6 - 3.9
- In reprorin heuristic, allow for having multiple accessions since now there is
  `-g all` grouping ([#508][])
- For BIDS, produce a singular `scans.json` at the top level, and not one per
  sub/ses (generates too many identical files) ([#507][])


## Fixed

- Compatibility with DataLad 0.15.0. Minimal version is 0.13.0 now.
- Try to open top level BIDS .json files a number of times for adjustment,
  so in the case of competition across parallel processes, they just end up
  with the last one "winning over" ([#523][])
- Don't fail if etelemetry.get_project returns None ([#501][])
- Consistently use `n/a` for age/sex, also handle ?M for months ([#500][])
- To avoid crashing on unrelated derivatives files etc, make `find_files` to
  take list of topdirs (excluding `derivatives/` etc),
  and look for _bold only under sub-* directories ([#496][])
- Ensure bvec/bval files are only created for dwi output ([#491][])

## Removed

- In reproin heuristic, old hardcoded sequence renamings and filters ([#508][])


# [0.9.0] - 2020-12-23

Various improvements and compatibility/support (dcm2niix, datalad,
duecredit) changes.  Major change is placement of output files to the
target output directory during conversion.

## Added

- #454 zenodo referencing in README.rst and support for ducredit for
  heudiconv and reproin heuristic
- #445 more tutorial references in README.md

## Changed

- [#485][] placed files during conversion right away into the target
  directory (with a `_heudiconv???` suffix, renamed into ultimate target
  name later on), which avoids hitting file size limits of /tmp ([#481][]) and
  helped to avoid a regression in dcm2nixx 1.0.20201102
- [#477][] replaced `rec-<magnitude|phase>` with `part-<mag|phase>` now
```

(continues on next page)

(continued from previous page)

```
  hat BIDSsupports the part entity
- [#473][] made default for CogAtlasID to be a TODO URL
- [#459][] made AcquisitionTime used for acq_time scans file field
- [#451][] retained sub-second resolution in scans files
- [#442][] refactored code so there is now heudiconv.main.workflow for
  more convenient use as a Python module

## Fixed

- minimal version of nipype set to 1.2.3 to guarantee correct handling
  of DWI files ([#480][])
- `heudiconvDCM*` temporary directories are removed now ([#462][])
- compatibility with DataLad 0.13 ([#464][])

## Removed

- #443 pathlib as a dependency (we are Python3 only now)


# [0.8.0] - 2020-04-15

## Enhancements

- Centralized saving of .json files.  Indentation of some files could
  change now from previous versions where it could have used `3`
  spaces. Now indentation should be consistently `2` for .json files
  we produce/modify ([#436][]) (note: dcm2niix uses tabs for indentation)
- ReproIn heuristic: support SBRef and phase data ([#387][])
- Set the "TaskName" field in .json sidecar files for multi-echo data
  ([#420][])
- Provide an informative exception if command needs heuristic to be
  specified ([#437][])

## Refactored

- `embed_nifti` was refactored into `embed_dicom_and_nifti_metadata`
  which would no longer create `.nii` file if it does not exist
  already ([#432][])

## Fixed

- Skip datalad-based tests if no datalad available ([#430][])
- Search heuristic file path first so we do not pick up a python
  module if name conflicts ([#434][])

# [0.7.0] - 2020-03-20

## Removed

- Python 2 support/testing

## Enhancement

- `-g` option obtained two new modes: `all` and `custom`. In case of `all`,
  all provided DICOMs will be treated as coming from a single scanning session.
  `custom` instructs to use `.grouping` value (could be a DICOM attribute or
  a callable)provided by the heuristic ([#359][]).
```

(continues on next page)

(continued from previous page)

```
- Stop before reading pixels data while gathering metadata from DICOMs ([#404][])
- reproin heuristic:
  - In addition to original "md5sum of the study_description" `protocols2fix`
    could now have (and applied after md5sum matching ones)
    1). a regular expression searched in study_description,
    2). an empty string as "catch all".
    This features could be used to easily provide remapping into reproin
    naming (documentation is to come to http://github.com/ReproNim/reproin)
    ([#425][])

## Fixed

- Use nan, not None for absent echo value in sorting
- reproin heuristic: case seqinfos into a list to be able to modify from
  overloaded heuristic ([#419][])
- No spurious errors from the logger upon a warning about `etelemetry`
  absence ([#407][])

# [0.6.0] - 2019-12-16

This is largely a bug fix.  Metadata and order of `_key-value` fields in BIDS
could change from the result of converting using previous versions, thus minor
version boost.
14 people contributed to this release -- thanks
[everyone](https://github.com/nipy/heudiconv/graphs/contributors)!

## Enhancement

- Use [etelemetry](https://pypi.org/project/etelemetry) to inform about most
  recent available version of heudiconv. Please set `NO_ET` environment variable
  if you want to disable it ([#369][])
- BIDS:
  - `--bids` flag became an option. It can (optionally) accept `notop` value
    to avoid creation of top level files (`CHANGES`, `dataset_description.json`,
    etc) as a workaround during parallel execution to avoid race conditions etc.
    ([#344][])
  - Generate basic `.json` files with descriptions of the fields for
    `participants.tsv` and `_scans.tsv` files ([#376][])
  - Use `filelock` while writing top level files. Use
    `HEUDICONV_FILELOCK_TIMEOUT` environment to change the default timeout value
    ([#348][])
  - `_PDT2` was added as a suffix for multi-echo (really "multi-modal")
    sequences ([#345][])
- Calls to `dcm2niix` would include full output path to make it easier to
  discern in the logs what file it is working on ([#351][])
- With recent [datalad]() (>= 0.10), created DataLad dataset will use
  `--fake-dates` functionality of DataLad to not leak data conversion dates,
  which might be close to actual data acquisition/patient visit ([#352][])
- Support multi-echo EPI `_phase` data ([#373][] fixes [#368][])
- Log location of a bad .json file to ease troubleshooting ([#379][])
- Add basic pypi classifiers for the package ([#380][])

## Fixed
- Sorting `_scans.tsv` files lacking valid dates field should not cause a crash
  ([#337][])
- Multi-echo files detection based number of echos ([#339][])
- BIDS
```

(continues on next page)

(continued from previous page)

```
  - Use `EchoTimes` from the associated multi-echo files if `EchoNumber` tag is
    missing ([#366][] fixes [#347][])
  - Tolerate empty ContentTime and/or ContentDate in DICOMs ([#372][]) and place
    "n/a" if value is missing ([#390][])
  - Do not crash and store original .json file is "JSON pretification" fails
    ([#342][])
- ReproIn heuristic
  - tolerate WIP prefix on Philips scanners ([#343][])
  - allow for use of `(...)` instead of `{...}` since `{}` are not allowed
    ([#343][])
  - Support pipolar fieldmaps by providing them with `_epi` not `_magnitude`.
    "Loose" BIDS `_key-value` pairs might come now after `_dir-` even if they
    came first before ([#358][] fixes [#357][])
- All heuristics saved under `.heudiconv/` under `heuristic.py` name, to avoid
  discrepancy during reconversion ([#354][] fixes [#353][])
- Do not crash (with TypeError) while trying to sort absent file list ([#360][])
- heudiconv requires nipype >= 1.0.0 ([#364][]) and blacklists `1.2.[12]` ([#375][])

# [0.5.4] - 2019-04-29

This release includes fixes to BIDS multi-echo conversions, the
re-implementation of queuing support (currently just SLURM), as well as
some bugfixes.

Starting today, we will (finally) push versioned releases to DockerHub.
Finally, to more accurately reflect on-going development, the `latest`
tag has been renamed to `unstable`.

## Added
- Readthedocs documentation ([#327][])

## Changed
- Update Docker dcm2niix to v.1.0.20190410 ([#334][])
- Allow usage of `--files` with basic heuristics. This requires
  use of `--subject` flag, and is limited to one subject. ([#293][])

## Deprecated

## Fixed
- Improve support for multiple `--queue-args` ([#328][])
- Fixed an issue where generated BIDS sidecar files were missing additional
  information - treating all conversions as if the `--minmeta` flag was
  used ([#306][])
- Re-enable SLURM queuing support ([#304][])
- BIDS multi-echo support for EPI + T1 images ([#293][])
- Correctly handle the case when `outtype` of heuristic has "dicom"
  before '.nii.gz'. Previously would have lead to absent additional metadata
  extraction etc ([#310][])

## Removed
- `--sbargs` argument was renamed to `--queue-args` ([#304][])

## Security


# [0.5.3] - 2019-01-12
```

(continues on next page)

(continued from previous page)

```
Minor hot bugfix release

## Fixed
- Do not shorten spaces in the dates while pretty printing .json

# [0.5.2] - 2019-01-04

A variety of bugfixes

## Changed
- Reproin heuristic: `__dup` indices would now be assigned incrementally
  individually per each sequence, so there is a chance to properly treat
  associate for multi-file (e.g. `fmap`) sequences
- Reproin heuristic: also split StudyDescription by space not only by ^
- `tests/` moved under `heudiconv/tests` to ease maintenance and facilitate
  testing of an installed heudiconv
- Protocol name will also be accessed from private Siemens
  csa.tProtocolName header field if not present in public one
- nipype>=0.12.0 is required now

## Fixed
- Multiple files produced by dcm2niix are first sorted to guarantee
  correct order e.g. of magnitude files in fieldmaps, which otherwise
  resulted in incorrect according to BIDS ordering of them
- Aggregated top level .json files now would contain only the fields
  with the same values from all scanned files. In prior versions,
  those files were not regenerated after an initial conversion
- Unicode handling in anonimization scripts

# [0.5.1] - 2018-07-05
Bugfix release

## Added
- Video tutorial / updated slides
- Helper to set metadata restrictions correctly
- Usage is now shown when run without arguments
- New fields to Seqinfo
  - series_uid
- Reproin heuristic support for xnat
## Changed
- Dockerfile updated to use `dcm2niix v1.0.20180622`
- Conversion table will be regenerated if heurisic has changed
- Do not touch existing BIDS files
  - events.tsv
  - task JSON
## Fixed
- Python 2.7.8 and older installation
- Support for updated packages
  - `Datalad` 0.10
  - `pydicom` 1.0.2
- Later versions of `pydicom` are prioritized first
- JSON pretty print should not remove spaces
- Phasediff fieldmaps behavior
  - ensure phasediff exists
  - support for single magnitude acquisitions

# [0.5] - 2018-03-01
```

(continues on next page)

(continued from previous page)

```
The first release after major refactoring:

## Changed
- Refactored into a proper `heudiconv` Python module
  - `heuristics` is now a `heudiconv.heuristics` submodule
  - you can specify shipped heuristics by name (e.g. `-f reproin`)
    without providing full path to their files
  - you need to use `--files` (not just positional argument(s)) if not
    using `--dicom_dir_templates` or `--subjects` to point to data files
    or directories with input DICOMs
- `Dockerfile` is generated by [neurodocker](https://github.com/kaczmarj/neurodocker)
- Logging verbosity reduced
- Increased leniency with missing DICOM fields
- `dbic_bids` heuristic renamed into reproin
## Added
- [LICENSE](https://github.com/nipy/heudiconv/blob/master/LICENSE)
  with Apache 2.0 license for the project
- [CHANGELOG.md](https://github.com/nipy/heudiconv/blob/master/CHANGELOG.md)
- [Regression testing](https://github.com/nipy/heudiconv/blob/master/tests/test_
↪regression.py) on real data (using datalad)
- A dedicated [ReproIn](https://github.com/repronim/reproin) project
  with details about ReproIn setup/specification and operation using
  `reproin` heuristic shipped with heudiconv
- [utils/test-compare-two-versions.sh](utils/test-compare-two-versions.sh)
  helper to compare conversions with two different versions of heudiconv
## Removed
- Support for converters other than `dcm2niix`, which is now the default.
  Explicitly specify `-c none` to only prepare conversion specification
  files without performing actual conversion
## Fixed
- Compatibility with Nipype 1.0, PyDicom 1.0, and upcoming DataLad 0.10
- Consistency with converted files permissions
- Ensured subject id for BIDS conversions will be BIDS compliant
- Re-add `seqinfo` fields as column names in generated `dicominfo`
- More robust sanity check of the regex reformatted .json file to avoid
  numeric precision issues
- Many other various issues

# [0.4] - 2017-10-15
A usable release to support [DBIC][] use-case
## Added
- more testing
## Changes
- Dockerfile updates (added pigz, progressed forward [dcm2niix][])
## Fixed
- correct date/time in BIDS `_scans` files
- sort entries in `_scans` by date and then filename

# [0.3] - 2017-07-10
A somewhat working release on the way to support [DBIC][] use-case
## Added
- more tests
- grouping of dicoms by series if provided
- many more features and fixes

# [0.2] - 2016-10-20
An initial release on the way to support [DBIC][] use-case
```

(continues on next page)

(continued from previous page)

```
## Added
- basic Python project assets (`setup.py`, etc)
- basic tests
- [datalad][] support
- dbic_bids heuristic
- `--dbg` command line flag to enter `pdb` environment upon failure
# Fixed
- Better Python3 support
- Better PEP8 compliance

# [0.1] - 2015-09-23

Initial version

---

## References
[DBIC]: http://dbic.dartmouth.edu
[datalad]: http://datalad.org
[dcm2niix]: https://github.com/rordenlab/dcm2niix
[#301]: https://github.com/nipy/heudiconv/issues/301
[#353]: https://github.com/nipy/heudiconv/issues/353
[#354]: https://github.com/nipy/heudiconv/issues/354
[#357]: https://github.com/nipy/heudiconv/issues/357
[#358]: https://github.com/nipy/heudiconv/issues/358
[#347]: https://github.com/nipy/heudiconv/issues/347
[#366]: https://github.com/nipy/heudiconv/issues/366
[#368]: https://github.com/nipy/heudiconv/issues/368
[#373]: https://github.com/nipy/heudiconv/issues/373
[#485]: https://github.com/nipy/heudiconv/issues/485
[#442]: https://github.com/nipy/heudiconv/issues/442
[#451]: https://github.com/nipy/heudiconv/issues/451
[#459]: https://github.com/nipy/heudiconv/issues/459
[#473]: https://github.com/nipy/heudiconv/issues/473
[#477]: https://github.com/nipy/heudiconv/issues/477
[#293]: https://github.com/nipy/heudiconv/issues/293
[#304]: https://github.com/nipy/heudiconv/issues/304
[#306]: https://github.com/nipy/heudiconv/issues/306
[#310]: https://github.com/nipy/heudiconv/issues/310
[#327]: https://github.com/nipy/heudiconv/issues/327
[#328]: https://github.com/nipy/heudiconv/issues/328
[#334]: https://github.com/nipy/heudiconv/issues/334
[#337]: https://github.com/nipy/heudiconv/issues/337
[#339]: https://github.com/nipy/heudiconv/issues/339
[#342]: https://github.com/nipy/heudiconv/issues/342
[#343]: https://github.com/nipy/heudiconv/issues/343
[#344]: https://github.com/nipy/heudiconv/issues/344
[#345]: https://github.com/nipy/heudiconv/issues/345
[#348]: https://github.com/nipy/heudiconv/issues/348
[#351]: https://github.com/nipy/heudiconv/issues/351
[#352]: https://github.com/nipy/heudiconv/issues/352
[#359]: https://github.com/nipy/heudiconv/issues/359
[#360]: https://github.com/nipy/heudiconv/issues/360
[#364]: https://github.com/nipy/heudiconv/issues/364
[#369]: https://github.com/nipy/heudiconv/issues/369
[#372]: https://github.com/nipy/heudiconv/issues/372
[#375]: https://github.com/nipy/heudiconv/issues/375
```

(continues on next page)

(continued from previous page)

```
[#376]: https://github.com/nipy/heudiconv/issues/376
[#379]: https://github.com/nipy/heudiconv/issues/379
[#380]: https://github.com/nipy/heudiconv/issues/380
[#387]: https://github.com/nipy/heudiconv/issues/387
[#390]: https://github.com/nipy/heudiconv/issues/390
[#404]: https://github.com/nipy/heudiconv/issues/404
[#407]: https://github.com/nipy/heudiconv/issues/407
[#419]: https://github.com/nipy/heudiconv/issues/419
[#420]: https://github.com/nipy/heudiconv/issues/420
[#425]: https://github.com/nipy/heudiconv/issues/425
[#430]: https://github.com/nipy/heudiconv/issues/430
[#432]: https://github.com/nipy/heudiconv/issues/432
[#434]: https://github.com/nipy/heudiconv/issues/434
[#436]: https://github.com/nipy/heudiconv/issues/436
[#437]: https://github.com/nipy/heudiconv/issues/437
[#462]: https://github.com/nipy/heudiconv/issues/462
[#464]: https://github.com/nipy/heudiconv/issues/464
[#480]: https://github.com/nipy/heudiconv/issues/480
[#481]: https://github.com/nipy/heudiconv/issues/481
[#487]: https://github.com/nipy/heudiconv/issues/487
[#489]: https://github.com/nipy/heudiconv/issues/489
[#491]: https://github.com/nipy/heudiconv/issues/491
[#496]: https://github.com/nipy/heudiconv/issues/496
[#500]: https://github.com/nipy/heudiconv/issues/500
[#501]: https://github.com/nipy/heudiconv/issues/501
[#507]: https://github.com/nipy/heudiconv/issues/507
[#508]: https://github.com/nipy/heudiconv/issues/508
[#523]: https://github.com/nipy/heudiconv/issues/523
```

6.5.3 Usage

`heudiconv` processes DICOM files and converts the output into user defined paths.

CommandLine Arguments

Example: heudiconv -d 'rawdata/{subject}' -o . -f heuristic.py -s s1 s2 s3

```
usage: heudiconv [-h] [--version]
                 [-d DICOM_DIR_TEMPLATE | --files [FILES ...]]
                 [-s [SUBJS ...]] [-c {dcm2niix,none}] [-o OUTDIR]
                 [-l LOCATOR] [-a CONV_OUTDIR] [--anon-cmd ANON_CMD]
                 [-f HEURISTIC] [-p] [-ss SESSION]
                 [-b [BIDSOPTION1 [BIDSOPTION2 ...]]] [--overwrite]
                 [--datalad] [--dbg]
                 [--command {heuristics,heuristic-info,ls,populate-templates,sanitize-
↪jsons,treat-jsons,populate-intended-for}]
                 [-g {studyUID,accession_number,all,custom}] [--minmeta]
                 [--random-seed RANDOM_SEED] [--dcmconfig DCMCONFIG]
                 [-q {SLURM,None}] [--queue-args QUEUE_ARGS]
```

Named Arguments

--version show program's version number and exit

-d, --dicom_dir_template Location of dicomdir that can be indexed with subject id {subject} and session {session}. Tarballs (can be compressed) are supported in addition to directory. All matching tarballs for a subject are extracted and their content processed in a single pass. If multiple tarballs are found, each is assumed to be a separate session and the –ses argument is ignored. Note that you might need to surround the value with quotes to avoid {. . . } being considered by shell

--files Files (tarballs, dicoms) or directories containing files to process. Cannot be provided if using –dicom_dir_template.

-s, --subjects List of subjects - required for dicom template. If not provided, DICOMS would first be "sorted" and subject IDs deduced by the heuristic.

-c, --converter Possible choices: dcm2niix, none

Tool to use for DICOM conversion. Setting to "none" disables the actual conversion step – useful for testing heuristics.

-o, --outdir Output directory for conversion setup (for further customization and future reference. This directory will refer to non-anonymized subject IDs.

-l, --locator Study path under outdir. If provided, it overloads the value provided by the heuristic. If –datalad is enabled, every directory within locator becomes a superdataset thus establishing a hierarchy. Setting to "unknown" will skip that dataset.

-a, --conv-outdir Output directory for converted files. By default this is identical to –outdir. This option is most useful in combination with –anon-cmd.

--anon-cmd Command to run to convert subject IDs used for DICOMs to anonymized IDs. Such command must take a single argument and return a single anonymized ID. Also see –conv-outdir.

-f, --heuristic Name of a known heuristic or path to the Python script containing heuristic.

-p, --with-prov Store additional provenance information. Requires python-rdflib.

-ss, --ses Session for longitudinal study_sessions. Default is None.

-b, --bids Possible choices: notop

Flag for output into BIDS structure. Can also take BIDS-specific options, e.g., –bids notop. The only currently supported options is "notop", which skips creation of top-level BIDS files. This is useful when running in batch mode to prevent possible race conditions.

--overwrite Overwrite existing converted files.

--datalad Store the entire collection as DataLad dataset(s). Small files will be committed directly to git, while large to annex. New version (6) of annex repositories will be used in a "thin" mode so it would look to mortals as just any other regular directory (i.e. no symlinks to under .git/annex). For now just for BIDS mode.

--dbg Do not catch exceptions and show exception traceback.

--command Possible choices: heuristics, heuristic-info, ls, populate-templates, sanitize-jsons, treat-jsons, populate-intended-for

Custom action to be performed on provided files instead of regular operation.

-g, --grouping Possible choices: studyUID, accession_number, all, custom

How to group dicoms (default: by studyUID).

--minmeta Exclude dcmstack meta information in sidecar jsons.

--random-seed	Random seed to initialize RNG.
--dcmconfig	JSON file for additional dcm2niix configuration.

Conversion submission options

-q, --queue	Possible choices: SLURM, None Batch system to submit jobs in parallel.
--queue-args	Additional queue arguments passed as a single string of space-separated Argument=Value pairs.

Support

All bugs, concerns and enhancement requests for this software can be submitted here: https://github.com/nipy/heudiconv/issues.

If you have a problem or would like to ask a question about how to use `heudiconv`, please submit a question to NeuroStars.org with a `heudiconv` tag. NeuroStars.org is a platform similar to StackOverflow but dedicated to neuroinformatics.

All previous `heudiconv` questions are available here: http://neurostars.org/tags/heudiconv/

Batch jobs

`heudiconv` can natively handle multi-subject, multi-session conversions although it will do these conversions in a linear manner, i.e. one subject and one session at a time. To speed up these conversions, multiple `heudiconv` processes can be spawned concurrently, each converting a different subject and/or session.

The following example uses SLURM and Singularity to submit every subjects' DICOMs as an independent `heudiconv` execution.

The first script aggregates the DICOM directories and submits them to `run_heudiconv.sh` with SLURM as a job array.

If using bids, the `notop` bids option suppresses creation of top-level files in the bids directory (e.g., `dataset_description.json`) to avoid possible race conditions. These files may be generated later with `populate_templates.sh` below (except for `participants.tsv`, which must be created manually).

```
#!/bin/bash

set -eu

# where the DICOMs are located
DCMROOT=/dicom/storage/voice
# where we want to output the data
OUTPUT=/converted/data/voice

# find all DICOM directories that start with "voice"
DCMDIRS=(`find ${DCMROOT} -maxdepth 1 -name voice* -type d`)

# submit to another script as a job array on SLURM
sbatch --array=0-`expr ${#DCMDIRS[@]} - 1` run_heudiconv.sh ${OUTPUT} ${DCMDIRS[@]}
```

The second script processes a DICOM directory with `heudiconv` using the built-in *reproin* heuristic.

```
#!/bin/bash
set -eu

OUTDIR=${1}
# receive all directories, and index them per job array
DCMDIRS=(${@:2})
DCMDIR=${DCMDIRS[${SLURM_ARRAY_TASK_ID}]}
echo Submitted directory: ${DCMDIR}

IMG="/singularity-images/heudiconv-latest-dev.sif"
CMD="singularity run -B ${DCMDIR}:/dicoms:ro -B ${OUTDIR}:/output -e ${IMG} --files /
↪dicoms/ -o /output -f reproin -c dcm2niix -b notop --minmeta -l ."

printf "Command:\n${CMD}\n"
${CMD}
echo "Successful process"
```

This script creates the top-level bids files (e.g., `dataset_description.json`)

```
#!/bin/bash
set -eu

OUTDIR=${1}
IMG="/singularity-images/heudiconv-latest-dev.sif"
CMD="singularity run -B ${OUTDIR}:/output -e ${IMG} --files /output -f reproin --
↪command populate-templates"

printf "Command:\n${CMD}\n"
${CMD}
echo "Successful process"
```

6.5.4 Heuristic

The heuristic file controls how information about the DICOMs is used to convert to a file system layout (e.g., BIDS). `heudiconv` includes some built-in heuristics, including ReproIn (which is great to adopt if you will be starting your data collection!).

However, there is a large variety of data out there, and not all DICOMs will be covered by the existing heuristics. This section will outline what makes up a heuristic file, and some useful functions available when making one.

Components

`infotodict(seqinfos)`

The only required function for a heuristic, *infotodict* is used to both define the conversion outputs and specify the criteria for scan to output association. Conversion outputs are defined as keys, a *tuple* consisting of three elements:

- a template path used for the basis of outputs
- *tuple* of output types. Valid types include *nii*, *nii.gz*, and *dicom*.
- *None* - a historical artifact (corresponds to some notion of `annotation_class` no living human is aware about)

Note: An example conversion key

```
('sub-{subject}/func/sub-{subject}_task-test_run-{item}_bold', ('nii.gz',
'dicom'), None)
```

The `seqinfos` parameter is a list of namedtuples which serves as a grouped and stacked record of the DICOMs passed in. Each item in *seqinfo* contains DICOM metadata that can be used to isolate the series, and assign it to a conversion key.

A function `create_key` is commonly defined by heuristics (internally) to assist in creating the key, and to be used inside `infotodict`.

A dictionary of {`conversion key`: `series_id`} is returned, where `series_id` is the 3rd (indexes as `[2]` or accessed as `.series_id` from `seqinfo`).

`create_key(template, outtype)`

A common helper function used to create the conversion key in `infotodict`.

`filter_files(fl)`

A utility function used to filter any input files.

If this function is included, every file found will go through this filter. Any files where this function returns `True` will be filtered out.

`filter_dicom(dcm_data)`

A utility function used to filter any DICOMs.

If this function is included, every DICOM found will go through this filter. Any DICOMs where this function returns `True` will be filtered out.

`infotoids(seqinfos, outdir)`

Further processing on `seqinfos` to deduce/customize subject, session, and locator.

A dictionary of {"locator": locator, "session": session, "subject": subject} is returned.

`grouping` string or `grouping(files, dcmfilter, seqinfo)`

Whenever `--grouping custom` (`-g custom`) is used, this attribute or callable will be used to inform how to group the DICOMs into separate groups. From original PR#359:

```
grouping = 'AcquisitionDate'
```

or:

```
def grouping(files, dcmfilter, seqinfo):
    seqinfos = collections.OrderedDict()
    ...
    return seqinfos  # ordered dict containing seqinfo objects: list of DICOMs
```

POPULATE_INTENDED_FOR_OPTS

Dictionary to specify options to populate the 'IntendedFor' field of the fmap jsons.

When a BIDS session has fmaps, they can automatically be assigned to be used for susceptibility distortion correction of other non-fmap images in the session (populating the 'IntendedFor' field in the fmap json file).

For this automated assignment, fmaps are taken as groups (_phase and _phasediff images and the corresponding _magnitude images; consecutive Spin-Echo images collected with opposite phase encoding polarity (pepolar case); etc.).

This is achieved by checking, for every non-fmap image in the session, which fmap groups are suitable candidates to correct for distortions in that image. Then, if there is more than one candidate (e.g., if there was a fmap collected at the beginning of the session and another one at the end), the user can specify which one to use.

The parameters that can be specified and the allowed options are defined in bids.py:

- 'matching_parameter': The imaging parameter that needs to match between the fmap and an image for the fmap to be considered as a suitable to correct that image. Allowed options are:
  - 'Shims': heudiconv will check the ShimSetting in the .json files and will only assign fmaps to images if the ShimSettings are identical for both.
  - 'ImagingVolume': both fmaps and images will need to have the same the imaging volume (the header affine transformation: position, orientation and voxel size, as well as number of voxels along each dimensions).
  - 'ModalityAcquisitionLabel': it checks for what modality (anat, func, dwi) each fmap is intended by checking the _acq- label in the fmap filename and finding corresponding modalities (e.g. _acq-fmri, _acq-bold and _acq-func will be matched with the func modality)
  - 'CustomAcquisitionLabel': it checks for what modality images each fmap is intended by checking the _acq- custom label (e.g. _acq-XYZ42) in the fmap filename, and matching it with: - the corresponding modality image _acq- label for modalities other than func (e.g. _acq-XYZ42 for dwi images) - the corresponding image _task- label for the func modality (e.g. _task-XYZ42)
  - 'Force': forces heudiconv to consider any fmaps in the session to be suitable for any image, no matter what the imaging parameters are.
- 'criterion': Criterion to decide which of the candidate fmaps will be assigned to a given file, if there are more than one. Allowed values are:
  - 'First': The first matching fmap.
  - 'Closest': The closest in time to the beginning of the image acquisition.

Note: Example:

```
POPULATE_INTENDED_FOR_OPTS = {
        'matching_parameters': ['ImagingVolume', 'Shims'],
        'criterion': 'Closest'
}
```

If POPULATE_INTENDED_FOR_OPTS is not present in the heuristic file, IntendedFor will not be populated automatically.

6.5.5 User Tutorials

Luckily(?), we live in an era of plentiful information. Below are some links to other users' tutorials covering their experience with `heudiconv`.

- YouTube tutorial by James Kent.
- Walkthrough by the Stanford Center for Reproducible Neuroscience.
- U of A Neuroimaging Core by Dianne Patterson.
- Sample Conversion: Coastal Coding 2019.
- A joined DataLad and HeuDiConv tutorial for reproducible fMRI studies.
- The ReproIn conversion workflow overview.
- Slides and recording of a ReproNim Webinar on `heudiconv`.

Caution: Some of these tutorials may not be up to date with the latest releases of `heudiconv`.

6.5.6 API Reference

BIDS

Handle BIDS specific operations

exception heudiconv.bids.**BIDSError**

class heudiconv.bids.**BIDSFile**(*entities: dict, suffix: str, extension: Optional[str]*)
as defined in https://bids-specification.readthedocs.io/en/stable/99-appendices/04-entity-table.html which might soon become machine readable order matters

classmethod parse(*filename: str*) → heudiconv.bids.BIDSFile
Parse the filename for BIDS entities, suffix and extension

heudiconv.bids.**HEUDICONV_VERSION_JSON_KEY = 'HeudiconvVersion'**
JSON Key where we will embed our version in the newly produced .json files

heudiconv.bids.**add_rows_to_scans_keys_file**(*fn: str*, *newrows: dict*) → None
Add new rows to the _scans file.

Parameters

- **fn** (*str*) – filename
- **newrows** (*dict*) – extra rows to add (acquisition time, referring physician, random string)

heudiconv.bids.**convert_sid_bids**(*subject_id: str*) → str
Shim for stripping any non-BIDS compliant characters within subject_id

Parameters **subject_id** (*string*) –

Returns **sid** – New subject ID

Return type string

heudiconv.bids.**find_compatible_fmaps_for_run**(*json_file: str*, *fmap_groups: dict*, *matching_parameters: list*) → dict
Finds compatible fmaps for a given run, for populate_intended_for. (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **json_file** (*str*) – path to the json file
- **fmap_groups** (*dict*) – key: prefix common to the group value: list of all fmap paths in the group
- **matching_parameters** (*list of str from AllowedFmapParameterMatching*) – matching_parameters that will be used to match runs

Returns **compatible_fmap_groups** – Subset of the fmap_groups which match json_file, according to the matching_parameters. key: prefix common to the group value: list of all fmap paths in the group

Return type dict

heudiconv.bids.**find_compatible_fmaps_for_session**(*path_to_bids_session: str*, *matching_parameters: list*) → Optional[dict]

Finds compatible fmaps for all non-fmap runs in a session. (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **path_to_bids_session** (*str*) – path to the session folder (or to the subject folder, if there are no sessions).
- **matching_parameters** (*list of str from AllowedFmapParameterMatching*) – matching_parameters that will be used to match runs

Returns **compatible_fmap** – Dict of compatible_fmaps_groups (values) for each non-fmap run (keys)

Return type dict

heudiconv.bids.**find_fmap_groups**(*fmap_dir: str*) → dict

Finds the different fmap groups in a fmap directory. By groups here we mean fmaps that are intended to go together (with reversed PE polarity, magnitude/phase, etc.)

Parameters **fmap_dir** (*str*) – path to the session folder (or to the subject folder, if there are no sessions).

Returns **fmap_groups** – key: prefix common to the group (e.g. no "dir" entity, "_phase"/"_magnitude", ...) value: list of all fmap paths in the group

Return type dict

heudiconv.bids.**find_subj_ses**(*f_name: str*) → tuple

Given a path to the bids formatted filename parse out subject/session

heudiconv.bids.**get_formatted_scans_key_row**(*dcm_fn: str | Path*) → list[str]

Parameters **dcm_fn** (*str*) –

Returns **row** – [ISO acquisition time, performing physician name, random string]

Return type list

heudiconv.bids.**get_key_info_for_fmap_assignment**(*json_file: str*, *matching_parameter: str*) → list

Gets key information needed to assign fmaps to other modalities. (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **json_file** (*str*) – path to the json file

- **matching_parameter** (*str in AllowedFmapParameterMatching*) – matching_parameter that will be used to match runs

Returns **key_info** – part of the json file that will need to match between the fmap and the other image

Return type list

heudiconv.bids.**get_shim_setting**(*json_file: str*) → Any
Gets the "ShimSetting" field from a json_file. If no "ShimSetting" present, return error

Parameters **json_file** (*str*) –

Returns

Return type str with "ShimSetting" value

heudiconv.bids.**maybe_na**(*val: Any*) → str
Return 'n/a' if non-None value represented as str is not empty

Primarily for the consistent use of lower case 'n/a' so 'N/A' and 'NA' are also treated as 'n/a'

heudiconv.bids.**populate_aggregated_jsons**(*path: str*) → None
Aggregate across the entire BIDS dataset `.jsons` into top level `.jsons`

Top level .json files would contain only the fields which are common to all `subject[/session]/type/*_modality.jsons`.

ATM aggregating only for `*_task*_bold.json` files. Only the task- and OPTIONAL _acq- field is retained within the aggregated filename. The other BIDS _key-value pairs are "aggregated over".

Parameters **path** (*str*) – Path to the top of the BIDS dataset

heudiconv.bids.**populate_bids_templates**(*path: str*, *defaults: Optional[dict] = None*) → None
Premake BIDS text files with templates

heudiconv.bids.**populate_intended_for**(*path_to_bids_session: str, matching_parameters: str | list[str], criterion: str*) → None
Adds the 'IntendedFor' field to the fmap .json files in a session folder. It goes through the session folders and for every json file, it finds compatible_fmaps: fmaps that have the same matching_parameters as the json file (e.g., same 'Shims').

If there are more than one compatible_fmaps, it will use the criterion specified by the user (default: 'Closest' in time).

Because fmaps come in groups (with reversed PE polarity, or magnitude/ phase), we work with fmap_groups.

Parameters

- **path_to_bids_session** (*str*) – path to the session folder (or to the subject folder, if there are no sessions).
- **matching_parameters** (*list of str from AllowedFmapParameterMatching*) – matching_parameters that will be used to match runs
- **criterion** (*str in ['First', 'Closest']*) – matching_parameters that will be used to decide which of the matching fmaps to use

heudiconv.bids.**sanitize_label**(*label: str*) → str
Strips any non-BIDS compliant characters within label

Parameters **label** (*string*) –

Returns **clean_label** – New, sanitized label

Return type string

heudiconv.bids.**save_scans_key**(*item: tuple*, *bids_files: list*) → None

Parameters

- **item** –
- **bids_files** (*list of str*) –

heudiconv.bids.**select_fmap_from_compatible_groups**(*json_file: str*, *compatible_fmap_groups: dict*, *criterion: str*) → Optional[str]

Selects the fmap that will be used to correct for distortions in json_file from the compatible fmap_groups list, based on the given criterion (Note: It is the responsibility of the calling function to make sure the arguments are OK)

Parameters

- **json_file** (*str*) – path to the json file
- **compatible_fmap_groups** (*dict*) – fmap_groups that are compatible with the specific json_file
- **criterion** (*str in ['First', 'Closest']*) – matching_parameters that will be used to decide which fmap to use

Returns **selected_fmap_key** – key from the compatible_fmap_groups for the selected fmap group

Return type str

heudiconv.bids.**treat_age**(*age: str | float | None*) → str | None

Age might encounter 'Y' suffix or be a float

heudiconv.bids.**tuneup_bids_json_files**(*json_files: list*) → None

Given a list of BIDS .json files, e.g.

Conversion

heudiconv.convert.**add_taskname_to_infofile**(*infofiles: str | list[str]*) → None

Add the "TaskName" field to json files with _task- entity in the name.

Note: _task- entity could be present not only in functional data but in many other modalities now.

Parameters **infofiles** (*list or str*) – json filenames or a single filename.

heudiconv.convert.**bvals_are_zero**(*bval_file: str*) → bool

Checks if all entries in a bvals file are zero (or 5, for Siemens files).

Parameters **bval_file** (*str*) – file with the bvals

Returns

Return type True if all are all 0 or 5; False otherwise.

heudiconv.convert.**convert**(*items: list[tuple[str, tuple[str, ...], list[str]]]*, *converter: str*, *scaninfo_suffix: str*, *custom_callable: Optional[Callable[[str, tuple[str, ...], list[str]], Any]]*, *with_prov: bool*, *bids_options: Optional[str]*, *outdir: str*, *min_meta: bool*, *overwrite: bool*, *symlink: bool = True*, *prov_file: Optional[str] = None*, *dcmconfig: Optional[str] = None*, *populate_intended_for_opts: Optional[PopulateIntendedForOpts] = None*) → None

Perform actual conversion (calls to converter etc) given info from heuristic's *infotodict*

heudiconv.convert.**convert_dicom**(*item_dicoms: list, bids_options: Optional[str], prefix: str, outdir: str, tempdirs: heudiconv.utils.TempDirs, _symlink: bool, overwrite: bool*) → None

Save DICOMs as output (default is by symbolic link)

Parameters

- **item_dicoms** (*list of filenames*) – DICOMs to save
- **bids_options** (*str or None*) – If not None then save to BIDS format. String may be empty or contain bids specific options
- **prefix** (*string*) – Conversion outname
- **outdir** (*string*) – Output directory
- **tempdirs** (*TempDirs instance*) – Object to handle temporary directories created TODO: remove
- **symlink** (*bool*) – Create softlink to DICOMs - if False, create hardlink instead.
- **overwrite** (*bool*) – If True, allows overwriting of previous conversion

heudiconv.convert.**nipype_convert**(*item_dicoms: list, prefix: str, with_prov: bool, bids_options: Optional[str], tmpdir: str, dcmconfig: Optional[str] = None*) → tuple

Converts DICOMs grouped from heuristic using Nipype's Dcm2niix interface.

Parameters

- **item_dicoms** (*list*) – DICOM files to convert
- **prefix** (*str*) – Heuristic output path
- **with_prov** (*bool*) – Store provenance information
- **bids_options** (*str or None*) – If not None then output BIDS sidecar JSONs String may contain bids specific options
- **tmpdir** (*str*) – Conversion working directory
- **dcmconfig** (*str, optional*) – JSON file used for additional Dcm2niix configuration

heudiconv.convert.**save_converted_files**(*res: nipype.pipeline.engine.nodes.Node, item_dicoms: list, bids_options: Optional[str], outtype: str, prefix: str, outname_bids: str, overwrite: bool*) → list

Copy converted files from tempdir to output directory.

Will rename files if necessary.

Parameters

- **res** (*Node*) – Nipype conversion Node with results
- **item_dicoms** (*list*) – Filenames of converted DICOMs
- **bids** (*list or None*) – If not list save to BIDS List may contain bids specific options
- **prefix** (*str*) –

Returns Converted BIDS files

Return type bids_outfiles

heudiconv.convert.**update_complex_name**(*metadata: dict*, *filename: str*) → str

Insert *_part-<mag|phase>* entity into filename if data are from a sequence with magnitude/phase part.

Parameters

- **metadata** (*dict*) – Scan metadata dictionary from BIDS sidecar file.
- **filename** (*str*) – Incoming filename

Returns **filename** – Updated filename with part entity added in appropriate position.

Return type str

heudiconv.convert.**update_multiecho_name**(*metadata: dict*, *filename: str*, *echo_times: list*) → str

Insert *_echo-<num>* entity into filename if data are from a multi-echo sequence.

Parameters

- **metadata** (*dict*) – Scan metadata dictionary from BIDS sidecar file.
- **filename** (*str*) – Incoming filename
- **echo_times** (*list*) – List of all echo times from scan. Used to determine the echo *number* (i.e., index) if field is missing from metadata.

Returns **filename** – Updated filename with echo entity added, if appropriate.

Return type str

heudiconv.convert.**update_uncombined_name**(*metadata: dict*, *filename: str*, *channel_names: list*) → str

Insert *_ch-<num>* entity into filename if data are from a sequence with "save uncombined".

Parameters

- **metadata** (*dict*) – Scan metadata dictionary from BIDS sidecar file.
- **filename** (*str*) – Incoming filename
- **channel_names** (*list*) – List of all channel names from scan. Used to determine the channel *number* (i.e., index) if field is missing from metadata.

Returns **filename** – Updated filename with ch entity added, if appropriate.

Return type str

DICOMS

class heudiconv.dicoms.**SeriesID**(*series_number*, *protocol_name*, *file_studyUID*)

file_studyUID
Alias for field number 2

protocol_name
Alias for field number 1

series_number
Alias for field number 0

heudiconv.dicoms.**compress_dicoms**(*dicom_list: list*, *out_prefix: str*, *tempdirs: heudiconv.utils.TempDirs*, *overwrite: bool*) → Optional[str]

Archives DICOMs into a tarball

Also tries to do it reproducibly, so takes the date for files and target tarball based on the series time (within the first file)

Parameters

- **dicom_list** (*list of str*) – list of dicom files
- **out_prefix** (*str*) – output path prefix, including the portion of the output file name before .dicom.tgz suffix
- **tempdirs** (TempDirs) – TempDirs object to handle multiple tmpdirs
- **overwrite** (*bool*) – Overwrite existing tarfiles

Returns **filename** – Result tarball

Return type str

heudiconv.dicoms.**create_seqinfo**(*mw: nibabel.nicom.dicomwrappers.Wrapper*, *series_files: list*, *series_id: str*) → heudiconv.utils.SeqInfo

Generate sequence info

Parameters

- **mw** (*Wrapper*) –
- **series_files** (*list*) –
- **series_id** (*str*) –

heudiconv.dicoms.**embed_dicom_and_nifti_metadata**(*dcmfiles: List[str], niftifile: str, infofile: Union[str, pathlib.Path], bids_info: Optional[Dict[str, Any]]*) → None

Embed metadata from nifti (affine etc) and dicoms into infofile (json)

niftifile should exist. Its affine's orientation information is used while establishing new *NiftiImage* out of dicom stack and together with *bids_info* (if provided) is dumped into json *infofile*

Parameters

- **dcmfiles** –
- **niftifile** –
- **infofile** –
- **bids_info** (*dict*) – Additional metadata to be embedded. *infofile* is overwritten if exists, so here you could pass some metadata which would overload (at the first level of the dict structure, no recursive fancy updates) what is obtained from nifti and dicoms

heudiconv.dicoms.**embed_metadata_from_dicoms**(*bids_options: Optional[str], item_dicoms: list, outname: str, outname_bids: str, prov_file: Optional[str], scaninfo: str, tempdirs: heudiconv.utils.TempDirs, with_prov: bool*) → None

Enhance sidecar information file with more information from DICOMs

Parameters

- **bids_options** –
- **item_dicoms** –
- **outname** –
- **outname_bids** –
- **prov_file** –
- **scaninfo** –
- **tempdirs** –

- **with_prov** –

heudiconv.dicoms.**get_datetime_from_dcm**(*dcm_data: pydicom.dataset.FileDataset*) → Optional[datetime.datetime]

Extract datetime from filedataset, or return None is no datetime information found.

Parameters **dcm_data** (*dcm.FileDataset*) – DICOM with header, e.g., as ready by pydicom.dcmread. Objects with __getitem__ and have those keys with values properly formatted may also work

Returns One of several datetimes that are related to when the scan occurred, or None if no datetime can be found

Return type Optional[datetime.datetime]

Notes

The following fields are checked in order

1. AcquisitionDate & AcquisitionTime (0008,0022); (0008,0032)
2. AcquisitionDateTime (0008,002A);
3. SeriesDate & SeriesTime (0008,0021); (0008,0031)

heudiconv.dicoms.**get_reproducible_int**(*dicom_list: list*) → int

Get integer that can be used to reproducibly sort input DICOMs, which is based on when they were acquired.

Parameters **dicom_list** (*list[str]*) – Paths to existing DICOM files

Returns An integer relating to when the DICOM was acquired

Return type int

Raises AssertionError

Notes

1. **When date and time for can be read (see *get_datetime_from_dcm()*), return** that value as time in seconds since epoch (i.e., Jan 1 1970).
2. **In cases where a date/time/datetime is not available (e.g., anonymization stripped this info), return** epoch + AcquisitionNumber (in seconds), which is AcquisitionNumber as an integer
3. If 1 and 2 are not possible, then raise AssertionError and provide message about missing information

Cases are based on only the first element of the dicom_list.

heudiconv.dicoms.**group_dicoms_into_seqinfos**(*files: list[str], grouping: str, file_filter: Optional[Callable[[str], Any]] = None, dcmfilter: Optional[Callable[[dcm.dataset.Dataset], Any]] = None, flatten: Literal[False, True] = False, custom_grouping: str | Callable[[list[str], Optional[Callable[[dcm.dataset.Dataset], Any]], type[SeqInfo]], dict[SeqInfo, list[str]]] | None = None*) → dict[Optional[str], dict[SeqInfo, list[str]]] | dict[SeqInfo, list[str]]

Process list of dicoms and return seqinfo and file group *seqinfo* contains per-sequence extract of fields from DICOMs which will be later provided into heuristics to decide on filenames

Parameters

- **files** (*list of str*) – List of files to consider
- **grouping** (*{'studyUID', 'accession_number', 'all', 'custom'}*) – How to group DICOMs for conversion. If 'custom', see *custom_grouping* parameter.
- **file_filter** (*callable, optional*) – Applied to each item of filenames. Should return True if file needs to be kept, False otherwise.
- **dcmfilter** (*callable, optional*) – If called on dcm_data and returns True, it is used to set series_id
- **flatten** (*bool, optional*) – Creates a flattened *seqinfo* with corresponding DICOM files. True when invoked with *dicom_dir_template*.
- **custom_grouping** (*str or callable, optional*) – grouping key defined within heuristic. Can be a string of a DICOM attribute, or a method that handles more complex groupings.

Returns

- **seqinfo** (*list of list*) – *seqinfo* is a list of info entries per each sequence (some entry there defines a key for *filegrp*)
- **filegrp** (*dict*) – *filegrp* is a dictionary with files grouped per each sequence

heudiconv.dicoms.**parse_private_csa_header**(*dcm_data: pydicom.dataset.Dataset*, *_public_attr: str*, *private_attr: str*, *default: Optional[str] = None*) → str

Parses CSA header in cases where value is not defined publicly

Parameters

- **dcm_data** (*pydicom Dataset object*) – DICOM metadata
- **public_attr** (*string*) – non-private DICOM attribute
- **private_attr** (*string*) – private DICOM attribute
- **(optional)** (*default*) – default value if private_attr not found

Returns **val (default** – private attribute value or default

Return type empty string)

heudiconv.dicoms.**validate_dicom**(*fl: str, dcmfilter: Optional[collections.abc.Callable[pydicom.dataset.Dataset, Any]]*) → Optional[tuple]

Parse DICOM attributes. Returns None if not valid.

Parsing

heudiconv.parser.**find_files**(*regex: str*, *topdir: list[str] | tuple[str, ...] | str = '.'*, *exclude: Optional[str] = None*, *exclude_vcs: bool = True*, *dirs: bool = False*) → Iterator[str]

Generator to find files matching regex

Parameters

- **regex** (*string*) –
- **exclude** (*string, optional*) – Matches to exclude

- **exclude_vcs** – If True, excludes commonly known VCS subdirectories. If string, used as regex to exclude those files (regex: *r'/.(?:git|gitattributes|svn|bzr|hg)(?:/|$)'*)
- **topdir** (*string or list, optional*) – Directory where to search
- **dirs** (*bool, optional*) – Either to match directories as well as files

heudiconv.parser.**get_extracted_dicoms**(*fl: collections.abc.Iterable[str]*) → collections.abc.ItemsView[Optional[str], list]

Given a collection of files and/or directories, list out and possibly extract the contents from archives.

Parameters **fl** – Files (possibly archived) to process.

Returns The absolute paths of (possibly newly extracted) files.

Return type ItemsView[str | None, list[str]]

Notes

For 'classical' heudiconv, if multiple archives are provided, they correspond to different sessions, so here we would group into sessions and return pairs *sessionid*, *files* with *sessionid* being None if no "sessions" detected for that file or there was just a single tarball in the list.

When contents of fl appear to be an unpackable archive, the contents are extracted into utils.TempDirs(f'heudiconvDCM') and the mode of all extracted files is set to 700.

When contents of fl are a list of unarchived files, they are treated as a single session.

When contents of fl are a list of unarchived and archived files, the unarchived files are grouped into a single session (key: None). If there is only one archived file, the contents of that file are grouped with the unarchived file. If there are multiple archived files, they are grouped into separate sessions.

heudiconv.parser.**get_study_sessions**(*dicom_dir_template: Optional[str], files_opt: Optional[list[str]], heuristic: ModuleType, outdir: str, session: Optional[str], sids: Optional[list[str]], grouping: str = 'studyUID'*) → dict[StudySessionInfo, list[str] | dict[SeqInfo, list[str]]]

Sort files or dicom seqinfos into study_sessions.

study_sessions put together files for a single session of a subject in a study. Two major possible workflows:

- if dicom_dir_template provided – doesn't pre-load DICOMs and just loads files pointed by each subject and possibly sessions as corresponding to different tarballs.
- if files_opt is provided, sorts all DICOMs it can find under those paths

Batch Queuing

heudiconv.queue.**clean_args**(*hargs: list*, *iterarg: str*, *iteridx: int*) → list

Filters arguments for batch submission.

Parameters

- **hargs** (*list*) – Command-line arguments
- **iterarg** (*str*) – Multi-argument to index (*subjects* OR *files*)
- **iteridx** (*int*) – *iterarg* index to submit

Returns **cmdargs** – Filtered arguments for batch submission

Return type list

Example

```
>>> from heudiconv.queue import clean_args
>>> cmd = ['heudiconv', '-d', '/some/{subject}/path',
...                     '-q', 'SLURM',
...                     '-s', 'sub-1', 'sub-2', 'sub-3', 'sub-4']
>>> clean_args(cmd, 'subjects', 0)
['heudiconv', '-d', '/some/{subject}/path', '-s', 'sub-1']
```

heudiconv.queue.**queue_conversion**(*queue: str*, *iterarg: str*, *iterables: int*, *queue_args: Optional[str] = None*) → None

Write out conversion arguments to file and submit to a job scheduler. Parses *sys.argv* for heudiconv arguments.

Parameters

- **queue** (*string*) – Batch scheduler to use
- **iterarg** (*str*) – Multi-argument to index (*subjects* OR *files*)
- **iterables** (*int*) – Number of *iterarg* arguments
- **queue_args** (*string (optional)*) – Additional queue arguments for job submission

Utility

Utility objects and functions

class heudiconv.utils.**File**(*name: str*, *executable: bool = False*)

Helper for a file entry in the create_tree/@with_tree

It allows to define additional settings for entries

class heudiconv.utils.**SeqInfo**(*total_files_till_now*, *example_dcm_file*, *series_id*, *dcm_dir_name*, *series_files*, *unspecified*, *dim1*, *dim2*, *dim3*, *dim4*, *TR*, *TE*, *protocol_name*, *is_motion_corrected*, *is_derived*, *patient_id*, *study_description*, *referring_physician_name*, *series_description*, *sequence_name*, *image_type*, *accession_number*, *patient_age*, *patient_sex*, *date*, *series_uid*, *time*)

TE
Alias for field number 11

TR
Alias for field number 10

accession_number
Alias for field number 21

date
Alias for field number 24

dcm_dir_name
Alias for field number 3

dim1
Alias for field number 6

dim2
Alias for field number 7

dim3
Alias for field number 8

dim4
Alias for field number 9

example_dcm_file
Alias for field number 1

image_type
Alias for field number 20

is_derived
Alias for field number 14

is_motion_corrected
Alias for field number 13

patient_age
Alias for field number 22

patient_id
Alias for field number 15

patient_sex
Alias for field number 23

protocol_name
Alias for field number 12

referring_physician_name
Alias for field number 17

sequence_name
Alias for field number 19

series_description
Alias for field number 18

series_files
Alias for field number 4

series_id
Alias for field number 2

series_uid
Alias for field number 25

study_description
Alias for field number 16

time
Alias for field number 26

total_files_till_now
Alias for field number 0

unspecified
Alias for field number 5

class heudiconv.utils.**StudySessionInfo**(*locator*, *session*, *subject*)

locator
Alias for field number 0

session
Alias for field number 1

subject
Alias for field number 2

class heudiconv.utils.**TempDirs**
A helper to centralize handling and cleanup of dirs

heudiconv.utils.**anonymize_sid**(*sid: AnyStr*, *anon_sid_cmd: str*) → AnyStr

Raises ValueError – if script returned an empty string (after whitespace stripping), or output with multiple words/lines.

heudiconv.utils.**assure_no_file_exists**(*path: str | Path*) → None
Check if file or symlink (git-annex?) exists, and if so – remove

heudiconv.utils.**clear_temp_dicoms**(*item_dicoms: list*) → None
Ensures DICOM temporary directories are safely cleared

heudiconv.utils.**create_file_if_missing**(*filename: str*, *content: str*) → bool
Create file if missing, so we do not override any possibly introduced changes

heudiconv.utils.**create_tree**(*path: str, tree: Union[Sequence[Tuple[Union[str, heudiconv.utils.File], Union[str, TreeSpec, dict]]], Mapping[str, Union[str, TreeSpec, dict]]], archives_leading_dir: bool = True*) → None
Given a list of tuples (name, load) or a dict create such a tree

if load is a tuple or a dict itself – that would create either a subtree or an archive with that content and place it into the tree if name ends with .tar.gz

heudiconv.utils.**docstring_parameter**(**sub*) → collections.abc.Callable[T, T]
Borrowed from https://stackoverflow.com/a/10308363/6145776

heudiconv.utils.**get_datetime**(*date: str*, *time: str*, ***, *microseconds: bool = True*) → str
Combine date and time from dicom to isoformat.

Parameters

- **date** (*str*) – Date in YYYYMMDD format.
- **time** (*str*) – Time in either HHMMSS.ffffff format or HHMMSS format.
- **microseconds** (*bool, optional*) – Either to include microseconds in the output

Returns **datetime_str** – Combined date and time in ISO format, with microseconds as if fraction was provided in 'time', and 'microseconds' was True.

Return type str

heudiconv.utils.**get_known_heuristic_names**() → list
Return a list of heuristic names present under heudiconv/heuristics

heudiconv.utils.**get_typed_attr**(*obj: Any, attr: str, _type: type[T], default: Optional[V] = None*) → T | V | None
Typecasts an object's named attribute. If the attribute cannot be converted, the default value is returned instead.

Parameters

- **obj** (*Object*) –
- **attr** (*Attribute*) –

- **_type** (*Type*) –
- **default** (*value, optional*) –

heudiconv.utils.**is_readonly**(*path: str*) → bool
Return True if it is a fully read-only file (dereferences the symlink)

heudiconv.utils.**json_dumps**(*json_obj: Any*, *indent: int = 2*, *sort_keys: bool = True*) → str
Unified (default indent and sort_keys) invocation of json.dumps

heudiconv.utils.**json_dumps_pretty**(*j: Any*, *indent: int = 2*, *sort_keys: bool = True*) → str
Given a json structure, pretty print it by colliding numeric arrays into a line.

If resultant structure differs from original – throws exception

heudiconv.utils.**load_heuristic**(*heuristic: str*) → module
Load heuristic from the file, return the module

heudiconv.utils.**load_json**(*filename: str | Path*, *retry: int = 0*) → Any
Load data from a json file

Parameters

- **filename** (*str or Path*) – Filename to load data from.
- **retry** (*int, optional*) – Number of times to retry opening/loading the file in case of failure. Code will sleep for 0.1 seconds between retries. Could be used in code which is not sensitive to order effects (e.g. like populating bids templates where the last one to do it, would make sure it would be the correct/final state).

Returns **data**

Return type dict

heudiconv.utils.**remove_prefix**(*s: str*, *pre: str*) → str
Remove prefix from the beginning of the string

Parameters

- **s** (*str*) –
- **pre** (*str*) –

Returns **s** – string with "pre" removed from the beginning (if present)

Return type str

heudiconv.utils.**remove_suffix**(*s: str*, *suf: str*) → str
Remove suffix from the end of the string

Parameters

- **s** (*str*) –
- **suf** (*str*) –

Returns **s** – string with "suf" removed from the end (if present)

Return type str

heudiconv.utils.**safe_copyfile**(*src: str*, *dest: str*, *overwrite: bool = False*) → None
Copy file but blow if destination name already exists

heudiconv.utils.**safe_movefile**(*src: str*, *dest: str*, *overwrite: bool = False*) → None
Move file but blow if destination name already exists

heudiconv.utils.**save_json**(*filename: str | Path*, *data: Any*, *indent: int = 2*, *sort_keys: bool = True*, *pretty: bool = False*) → None

Save data to a json file

Parameters

- **filename** (*str or Path*) – Filename to save data in.
- **data** (*dict*) – Dictionary to save in json file.
- **indent** (*int, optional*) –
- **sort_keys** (*bool, optional*) –
- **pretty** (*bool, optional*) –

heudiconv.utils.**set_readonly**(*path: str*, *read_only: bool = True*) → int

Make file read only or writeable while preserving "access levels"

So if file was not readable by others, it should remain not readable by others.

Parameters

- **path** (*str*) –
- **read_only** (*bool, optional*) – If True (default) - would make it read-only. If False, would make it writeable for levels where it is readable

heudiconv.utils.**slim_down_info**(*j: dict*) → dict

Given an aggregated info structure, removes excessive details

Such as CSA fields, and SourceImageSequence which on Siemens files could be huge and not providing any additional immediately usable information. If needed, could be recovered from stored DICOMs

heudiconv.utils.**treat_infofile**(*filename: str*) → None

Tune up generated .json file (slim down, pretty-print for humans).

heudiconv.utils.**update_json**(*json_file: str | Path, new_data: dict[str, Any], pretty: bool = False*) → None

Adds a given field (and its value) to a json file

Parameters

- **json_file** (*str or Path*) – path for the corresponding json file
- **new_data** (*dict*) – pair of "key": "value" to add to the json file
- **pretty** (*bool*) – argument to be passed to save_json

Python Module Index

h

Index

U

V