**MEG Data Analysis Pipeline Outline**

Version 2.0

June 13, 2024

MEG analysis pipeline (this outline was put together based on what was previously done on pilot data and recent discoveries that not all MEG data has the 29.5Hz noise):

Pipeline A and B can be done concurrently to save time.

**MEG RAW DATA**

Pipeline B

Pipeline A

First, define bad channels.

- get bad channels from manual run-sheet

- To check raw data for bad channels, you first apply low pass filter to data to remove high frequencies if cHPI coils and other interferences. (55Hz bandpass).

Because not all raw MEG data contains the 29.5Hz artefact, run **vib\_artifact\_test\_1.py** first to check if the artefact is present in the data. Can ask Maggie to look at the generated figures to see if the 29.5Hz noise is present.

At this stage, run the script **preprocessing\_2.py or preprocessing\_no\_eSSS\_2.py**. Both preprocessing scripts also run tSSS and movement compensation on the data.

Run recon-all via Freesurfer (can be run separately or done together during the fMRI data prep stage).

Coreg:

Do by script or manual GUI

Bem solution file

Remove noise (“Preprocessing”):

If the 29.5Hz noise is present in the data, run **preprocessing\_2.py.**

If the 29.5Hz noise is NOT present in the data, run **preprocessing\_no\_eSSS\_2.py**.

At this stage, the scripts above do:

-Movement compensation.

-maxwell\_filter(tSSS).

-eSSS (if the scripts contain eSSS to remove the 29.5Hz noise)

Source Space

- setup\_sourse.py

-write\_source\_spaces()

After preprocessing, do ICA:

- remove blinking artefact) (run script **ICA\_3.py**)

After ICA, do **epoching\_4.py** (we use tSSS’ed data for this stage)/averaging conditions

* Note: Account for the image delay at this stage (80ms)

Can also run event ID script for visualizing event triggers but optional.

After epoching/averaging, do

forward model and inverse solution.

(no formal scripts yet as we did not reach this stage for the pilot data analysis)

https://mne.tools/stable/auto\_examples/inverse/mixed\_source\_space\_inverse.html

https://mne.tools/stable/generated/mne.make\_forward\_solution.html

Before proceeding to forward mode/inverse solution, make sure you have:

a. -trans.fif file from coregistration

b. source space

c. the BEM surfaces

https://mne.tools/stable/auto\_tutorials/forward/index.html

Scripts to be found on Maggie’s githug: [**https://github.com/mdclarke/Doesburg\_religion**](https://github.com/mdclarke/Doesburg_religion)

1. vib\_artifact\_test\_1.py

2. preprocessing\_2.py or preprocessing\_no\_eSSS\_2.py (depending on whether the 29.5Hz artefact is present in the MEG data)

3. ICA\_3.py

4. epoching\_4.py

\*\*\*PLEASE NOTE: we did not do eSSS when we analyzed pilot data subject 006. The 29.5Hz data is also not present in every participant raw MEG data so we should check whether the noise is actually present before running eSSS.

Other notes: It \*may\* be recommended to only run eSSS **on empty room**, **resting state, and movie data** to get rid of the 29.5Hz noise. Because we average and chunk task data together, the 29.5Hz noise may not be as big of a problem so we \*may\* not need to run eSSS on task data.

Extra notes about running eSSS:

* **Do not run tSSS twice.** Double check each script for potential duplicate functions to avoid running the same code twice.
* To run eSSS, you need the empty room file.
* The esss function is the argument extended\_Proj()
* Running Tsss, esss, movement compensation all in one step is ok. Do it once. Don’t run TSSS twice or you will run into problems.

## Example of what eSSS looks like (this is just for reference. Just use Maggie’s github script posted online. I copy-pasted the code below to help me quickly identify where we do eSSS)

## compute erm projectors to use for eSSS

## run this first before using the extended\_proj argument

erm\_proj = mne.compute\_proj\_raw(erm, meg='combined')

# plot the combined projectors

mne.viz.plot\_projs\_topomap(erm\_proj, colorbar=True,

info=erm.info)

# perform eSSS

esss = maxwell\_filter(raw, calibration=fc, cross\_talk=ct, st\_duration=10,

**extended\_proj=erm\_proj**)

**Pipeline A for analyzing MEG data:**

First, get “ct\_sparse.fif” and “sss\_cal.daf” files from the MEG techs.

Step 1.

1. **Defining Bad Channels During Data Collection:** a. **Manual Run-sheet:** During the MEG data collection, the MEG technician records channels suspected to be bad on a paper runsheet. This could include channels showing consistently noisy or abnormal signals. This is done by looking at the bad channel recorded on the paper runsheet.
2. **Checking Raw Data for Bad Channels Post-Collection:** a. **Apply a Low-Pass Filter to the Data:** To identify bad channels, one method is to apply a low-pass filter (e.g., a 55Hz bandpass filter) to remove high-frequency noise, such as cHPI coils and other interferences. By observing the filtered data, it might help highlight channels with irregular or problematic signals.
3. **Check if the 29.5Hz noise is present in the MEG data**. Not all raw MEG data contains the 29.5Hz. To check whether the 29.5Hz noise is present, run **vib\_artifact\_test\_1.py.**

Again, after you have done vib\_artifact\_test\_1.py, do:

2. preprocessing\_2.py **or** preprocessing\_no\_eSSS\_2.py (depending on whether the 29.5Hz artefact is present in the MEG data)

3. ICA\_3.py

4. epoching\_4.py

(the scripts are numbered accordingly)

**Pipeline B.**

1. **Run recon-all by free-surfer (separately or done together done during the preprocessing MRI stage).** After you have maxwell-filtered (aka. tSSS) you can run recon-all on the data. However, Leila was able to run recon-all on the data when she completed the fMRI prep/processing stage. The fMRI data processing stage will be outlined below instead:

**In order to use preprocessed data we need to prepare fMRI data to be used in the processing of data.**

1. Get DICOM file from the MRI tech lab
2. Convert them into NIFTI via MRIcroGI
3. Access Compute Canada and create a folder to work with MRI data (Upload NIFTI files to CC)
4. Sort them in BIDS. Example, path to my folder /project/6019337/leila3/pilot\_relig\_data/fmri\_raw\_pilot/data
5. Get a free licence at FreeSurfer License and upload them to your folder: <https://surfer.nmr.mgh.harvard.edu/fswiki/License>  
   <https://surfer.nmr.mgh.harvard.edu/registration.html>
6. Create some files or copy from my directory in CC. All of these files are required for MRI preprocessing. Example from my directory: /project/6019337/leila3/pilot\_relig\_data/fmri\_raw\_pilot/data

**List of files:**

**fmriprep-20.2.1.simg  
README**

**license.txt**

**pilot-sub-01.slurm.sh** ( you will have to either edit this file for each subject manually or create a loop.)  
**participants.tsv** ( you will have to edit it according to your “subject”)   
**dataset\_description.json** ( you will have to edit it according to your “subject”)

**event file .tsv** ( at this stage we need csv/log file from PsychoPy. **FYI: event file .tsv is required for further MRI analysis, not for preprocessing)**:

Before converting it to tsv, make sure you recalculate csv files with the right “timing”, it has to be started from 0.0 . Psychopy provides its own timing, so we just need to recalculate. The script for it is recalculator.py. Path to this this script /project/6019337/leila3/pilot\_relig\_data/fmri\_raw\_pilot

Then create foe each run a separate file manually (clean everything except block1, then block2, block3, ex:

\*\*pilot6\_recalculated\_run\_1.csv

pilot6\_recalculated\_run\_2.csv

pilot6\_recalculated\_run\_3.csv\*\*

After convert csv recalculated into tsv

Script for tsv is in “fmri\_raw\_pilot” folder.

Path to the script /project/6019337/leila3/pilot\_relig\_data/fmri\_raw\_pilot - tsv\_converter.py

How to run the script:

python3 tsv\_converter.py ./data/pilot1.csv ./data\_result -s

**BIDS event.tsv format:** BIDS event.tsv files typically have at least three columns: 'onset', 'duration', and 'trial\_type'. The 'onset' column represents the time (in seconds) from the start of the acquisition of the first volume in the corresponding imaging data file. The 'duration' column represents the duration (in seconds) of the event or trial. The 'trial\_type' column represents the type of event or trial.

After converting tsv into txt (3 column format for FSL), see the next step below

1. FSL 3-column format ( you will need to work FSL tool and FEAT GUI - **FYI: FSL 3-column format is required for further MRI analysis, not for preprocessing**)  
   FSL requires event files in a 3-column format, where the first column is the onset time (in seconds), the second column is the duration (in seconds), and the third column is the amplitude of the response (usually set to 1).  
     
   Convert the BIDS event.tsv file to the FSL 3-column format: You can do this using a python script.   
     
   The script would read the BIDS event.tsv file, select the 'onset' and 'duration' columns, add a new column for the amplitude (usually set to 1), and then write out a new file in the FSL 3-column format. You would typically do this separately for each 'trial\_type', resulting in a separate FSL 3-column file for each trial type.  
     
   You can see an example here: https://www.fmrib.ox.ac.uk/primers/intro\_primer/ExBox11/IntroBox11.html

Here is the script to convert event.tsv to FSL 3-column format. You can find the script in CC: /project/6019337/leila3/pilot\_relig\_data/fmri\_raw\_pilot

File name “fsl\_3.py”

1. Once you have all listed filed below except event file .tsv and FSL 3-column format , you are ready for preprocessing.

Submitting the script command sbatch ( to run the script)

Ex: sbatch pilot-sub-01.slurm.sh

As a result: fMRIprep will be preprocessed and converted it to the MNI space.  
Separately, Freesurfer will run recon-all (it is part of fMRIprep by default). This will be used for the MRI-MEG.

**—----------------------------------**

MRI-MEG co-registration (can be done by running a script or using GUI)

**The path to the script: /project/6019337/leila3**

**MRI\_MEG\_coregistration.py (what we used for pilot 6)**

**Script**

# Author: Jon Houck <jon.houck@gmail.com>

# Guillaume Favelier <guillaume.favelier@gmail.com>

#

# License: BSD-3-Clause

import numpy as np

import mne

from mne.coreg import Coregistration

from mne.io import read\_info

data\_path = '/project/6019337/leila3/pilot\_relig\_data/fmri\_raw\_pilot/preprocessed\_fmri\_23.1.3/sourcedata/'

subjects\_dir = data\_path + 'freesurfer'

subject = "sub-01"

fname\_raw = '/project/6019337/leila3/pilot\_relig\_data/meg\_raw\_pilot/sub-01/task\_pilot1.fif'

info = read\_info(fname\_raw)

plot\_kwargs = dict(

subject=subject,

subjects\_dir=subjects\_dir,

surfaces="head-dense",

dig=True,

eeg=[],

meg="sensors",

show\_axes=True,

coord\_frame="meg",

)

view\_kwargs = dict(azimuth=45, elevation=90, distance=0.6, focalpoint=(0.0, 0.0, 0.0))

fiducials = "estimated" # get fiducials from fsaverage

coreg = Coregistration(info, subject, subjects\_dir, fiducials=fiducials)

fig = mne.viz.plot\_alignment(info, trans=coreg.trans, \*\*plot\_kwargs)

**Bem files (scripts with tutorial to get these files):**

First, please run **mne.bem.make\_watershed\_bem** <https://mne.tools/stable/generated/mne.bem.make_watershed_bem.html#mne.bem.make_watershed_bem>

Define subject, subjects\_dir

This function should create the BEM surfaces in the bem/watershed directory of the subject's FreeSurfer directory.

Then please run **mne.bem.make\_scalp\_surfaces**

https://mne.tools/stable/generated/mne.bem.make\_scalp\_surfaces.html

Define subject, subjects\_dir

The output should look like this:/project/6019337/databases/camcan872\_fs/sub-CC620821/bem/

Or check folder: /project/6019337/leila3/pilot\_relig\_data/fmri\_raw\_pilot/preprocessed\_fmri\_23.1.3/sourcedata/freesurfer/sub-01/bem

Later on, you will need **https://mne.tools/stable/generated/mne.setup\_source\_space.html#mne.setup\_source\_space**

with spacing spacing="oct6"

- - - - - - -

Ready scripts to use to get the bem files:

Path to the scripts: /**project/6019337/leila3/bem**

generate\_watershed.py

scalp\_surfaces.py

\*\*Run the bem generation script:\*\*

python3 {script\_path\_with\_name}.py (be aware that the the file path and name may vary)

Inside the script you need to specify the path to the dir with subjects

**GUI coregistration**

- \*\*Or instead of running script for\*\* MRI\_MEG\_coregistration.py \*\*you can just easily open raw fif file in GUI with the command to do everything manually. You can use raw fif file or tsss data (maxwell filter). It doesn’t matter. The out put will be a trans file. The same if we do for MRI-MEG-coreg.\*\*

https://mne.tools/dev/generated/mne.gui.coregistration.html

mne coreg --subjects-dir=~/Desktop/coreg --subject=sub-01

—-----------------

**Bem Solution**<https://mne.tools/stable/generated/mne.make_bem_solution.html#mne.make_bem_solution>

Script - Leila ran it locally and used command read\_bem\_surfaces<https://mne.tools/stable/generated/mne.read_bem_surfaces.html#mne.read_bem_surfaces>

Script

import mne

surface\_file\_path = './sub-01/bem/sub-01-head-dense.fif'

surfs = mne.read\_bem\_surfaces(surface\_file\_path)

bem\_sol = mne.make\_bem\_solution(surfs)

bem\_sol.save('path name sub number'-5120-bem-sol.fif)

—------------

**Setup Source Space**  
  
<https://mne.tools/stable/generated/mne.setup_source_space.html#mne.setup_source_space>

Set up bilateral hemisphere surface-based source space with subsampling. You will not see the output, but you set up the source space for each hemisphere that will be used for tsss data (maxwell filter).

Script

import mne

subject = 'sub-01'

subjects\_dir = './'

result = mne.setup\_source\_space(subject, spacing='oct6', surface='white', subjects\_dir=subjects\_dir, add\_dist=True, n\_jobs=None, verbose=None)

print(result)

To save output  
  
save as source space file ( result.save (path)

write\_source\_spaces()

1. **Running Cog-Reg by script OR using manual GUI interface**

Coregistration

**What this stage does:** The coregistration process aims to spatially align the structural MRI-derived brain model (which includes cortical surfaces) with the MEG sensor positions. This alignment is essential for accurately localizing brain activity recorded by MEG sensors to specific anatomical structures within an individual's brain.

**(note: we need weighted T1 for this stage. Ideally we also need to run recon-all before getting to this stage.)**

1. With coregistration, you can start with raw MEG data (i.e. raw MEG data that doesn’t have its noise removed yet).

2. Do Coreg. Can be done by running the script from here <https://mne.tools/stable/auto_tutorials/forward/25_automated_coreg.html#initial-fit-with-fiducials> or manually using GUI <https://mne.tools/stable/generated/mne.gui.coregistration.html>. Both accomplishes the same task, but doing it manually allows you greater control/precision of fit.

When you have finished with coreg, you get the output file of a transformation matrix (a -trans.fif file that is basically a matrix of some numbers)

If manually aligning points, use **mne.viz.plot\_alignmen**t to look at the output. You will see what you see in the GUI (see how points are aligned). We are interested in how the white spots or white “spider web” fits on the scalp of the head (the white spots are the digitalization points, can look like a white spider web in the beginning).

If a white point is off (too far from head), this is a bad point that you can omit in GUI (or coreg script).

If you are using the coreg script, you can also use **mne.viz.plot\_alignment** to check output.

When using the GUI manually, set ICP to 5 (Default is 20) and see what happens. This number is not absolute and can be changed.

The white points are the digitization points. The blue colours are the sensors. We are interested in the white points. The goal is fit all the white points as close as to the surface of the scalp as possible. The pink dots are the 5 coils.

It is optional and not always necessary to omit “bad points”.

Sample image from sub-01:

A screenshot of a computer

Description automatically generated

Figure 1: Fitting the white points on the head. What it is supposed to look like

A screenshot of a computer

Description automatically generated

3. After we get the -trans.fif file, we can run “mne.make\_bem\_solution: <https://mne.tools/stable/generated/mne.make_bem_solution.html>

4. After this, we set up source space ( we need bem file to do this). This makes a grid inside the BEM where sources can fall.

For parameters, you need to define subject, spacing=“oct6”<https://mne.tools/stable/generated/mne.setup_source_space.html#mne.setup_source_space>

5. Get BEM solution file (supposed to be able to get from GUI). If not, the BEM solution file can be obtained by running this separately <https://mne.tools/stable/generated/mne.make_bem_solution.html>

6. Do Source space

**If running Coreg by script and not via manual GUI:**

**COREG code (modified from** [**https://mne.tools/stable/auto\_tutorials/forward/25\_automated\_coreg.html#initial-fit-with-fiducials**](https://mne.tools/stable/auto_tutorials/forward/25_automated_coreg.html#initial-fit-with-fiducials)**): the following is where we do coregistration “automatically” by running a script. Please note if you use this method, the coordinates may not fit as accurately and precisely as you would get if you were to manually fit points with using the GUI method.**

**The Co-Reg script below is what NY used to process pilot 6, but we ended up using Leila’s CoReg script instead. I saved the script below in case anyone find it useful.**

import numpy as np

import mne

from mne.coreg import Coregistration

from mne.io import read\_info

subjects\_dir = "/path"

subject = "sub-01"

fname\_raw = "path/sub\_01-1\_raw\_ sss\_meg.fif"

info = read\_info(fname\_raw)

plot\_kwargs = dict(

subject=subject,

subjects\_dir=subjects\_dir,

surfaces="head-dense",

dig=True,

eeg=[],

meg="sensors",

show\_axes=True,

coord\_frame="meg",

)

view\_kwargs = dict(azimuth=45, elevation=90, distance=0.6, focalpoint=(0.0, 0.0, 0.0))

# initial fit with fiducials

fiducials = "estimated" # get fiducials from fsaverage

coreg = Coregistration(info, subject="sub-01", subjects\_dir="pilot\_data", fiducials=fiducials)

fig = mne.viz.plot\_alignment(info, trans=coreg.trans, \*\*plot\_kwargs)

# Refining with ICP

coreg.fit\_icp(n\_iterations=6, nasion\_weight=2.0, verbose=True)

fig = mne.viz.plot\_alignment(info, trans=coreg.trans, \*\*plot\_kwargs)

# Omit bad points

# coreg.omit\_head\_shape\_points(distance=5.0 / 1000) # distance is in meters

# adjust accordingly

# Final Coregistration fit

coreg.fit\_icp(n\_iterations=20, nasion\_weight=10.0, verbose=True)

fig = mne.viz.plot\_alignment(info, trans=coreg.trans, \*\*plot\_kwargs)

mne.viz.set\_3d\_view(fig, \*\*view\_kwargs)

dists = coreg.compute\_dig\_mri\_distances() \* 1e3 # in mm

print(

f"Distance between HSP and MRI (mean/min/max):\n{np.mean(dists):.2f} mm "

f"/ {np.min(dists):.2f} mm / {np.max(dists):.2f} mm"

)

# save file (the resulting trans matrix)

# MNE naming conventions. All trans files should end **with -trans.fif, -trans.fif.gz, \_trans.fif or \_trans.fif.gz**

mne.write\_trans('/Applications/freesurfer/7.4.1/subjects/pilot\_data/sub-01/sub-01-trans.fif', coreg.trans)

**MISC (not in the official pipeline but useful scripts):**

SCRIPT FOR PLOTTING EVENT IDS BELOW

Helpful tidbit: Depending on which run you are running (1, 2, or 3), you may need to comment out the irrelevant block onset/offset.

import os  
import numpy as np  
import mne  
  
# sample\_data\_folder = '~/Desktop/mne\_events'

sample\_data\_folder = '/Applications/freesurfer/7.4.1/subjects/pilot\_data/sub-02/meg'  
sample\_data\_raw\_file = 'sub\_03\_run\_1\_raw.fif'

raw = mne.io.read\_raw\_fif(sample\_data\_raw\_file, verbose=False)  
  
events = mne.find\_events(raw, min\_duration=0.05)  
  
event\_dict = {  
# Group  
'God Start': 111,  
'God End': 112,  
'Inan Start': 121,  
'Inan End': 122,  
'Human Start': 131,  
'Human End': 132,  
'Super Start': 141,  
'Super End': 142,  
# Block  
'Block 1 Onset': 151,  
'Block 1 Offset': 152,  
# 'Block 2 Onset': 161,  
# 'Block 2 Offset': 162,  
#'Block 3 Onset': 171,  
#'Block 3 Offset': 172,  
# Question  
'Source of MORALITY? Onset': 41,  
'Source of MORALITY? Offset': 42,  
'Extent of CONNECTION? Onset': 51,  
'Extent of CONNECTION? Offset': 52,  
'Extent of MIND? Onset': 61,  
'Extent of MIND? Offset': 62  
}

fig = mne.viz.plot\_events(events, sfreq=[raw.info](http://raw.info/)['sfreq'],  
                          first\_samp=raw.first\_samp, event\_id=event\_dict)  
fig.subplots\_adjust(right=0.7)  # make room for legend

# file name convention: sub\_04\_run\_3\_event\_id # change number of subject and run # accordingly

**Script for making a standalone dataset description that does not depend on whether your data is in BIDS format:**

Please note, we did not use the following script during pilot data analysis. This is new and modified from the script that was used when we were doing BIDS. I have posted this modified script in case it may come handy in the future.

import os

import json

from pprint import pprint

# Define the path where the dataset\_description.json will be saved

dataset\_description\_path = "your\_path\_here" # Replace with your actual path

# Your task name

task = "image" # Replace with your actual task name

# Metadata for dataset\_description.json. All placeholder values for now

dataset\_description = {

"Name": task,

"Authors": ["Author1", "Author2"],

"How\_to\_acknowledge": """If you reference this dataset in a publication, please acknowledge its authors.""",

"Acknowledgements": "Thanks to everyone who contributed.",

"Data\_license": "CC0",

"Ethics\_approvals": ["Ethics approval code"],

"Funding": ["Funding information"],

"References\_and\_links": ["Link to paper"],

"DOI": "DOI number",

}

# Save dataset\_description to JSON file

with open(os.path.join(dataset\_description\_path, "dataset\_description.json"), "w") as f:

json.dump(dataset\_description, f, indent=4)

# Path to the generated dataset\_description.json

desc\_json\_path = os.path.join(dataset\_description\_path, "dataset\_description.json")

with open(desc\_json\_path, encoding="utf-8-sig") as fid:

pprint(json.loads(fid.read()))

# This should be very similar to the `ds000248 dataset\_description.json`\_!

Script for making a dataset description if your data is already organized into BIDS.

Please note, the following script was done in the context of having your data files arranged in BIDS first. I do not know if the following script would work if the data is not already in BIDS.

The “how to acknowledge” section and reference links are taken from the template script available from the MNE python website.

import os

import json from pprint

import pprint from mne\_bids

import make\_dataset\_description, BIDSPath

# %%

# It is also generally a good idea to add a description of your dataset,

# see the `BIDS dataset\_description.json definition`\_ for more information.

#how\_to\_acknowledge = """\

#If you reference this dataset in a publication, please acknowledge its \

#authors and cite MNE papers: A. Gramfort, M. Luessi, E. Larson, D. Engemann, \

#D. Strohmeier, C. Brodbeck, L. Parkkonen, M. Hämäläinen, \

#MNE software for processing MEG and EEG data, NeuroImage, Volume 86, \

#1 February 2014, Pages 446-460, ISSN 1053-8119 \

#and \

#A. Gramfort, M. Luessi, E. Larson, D. Engemann, D. Strohmeier, C. Brodbeck, \

#R. Goj, M. Jas, T. Brooks, L. Parkkonen, M. Hämäläinen, MEG and EEG data \

#analysis with MNE-Python, Frontiers in Neuroscience, Volume 7, 2013, \

I#SSN 1662-453X"""

# you need the bids path root to run the following; originally I only ran this script after doing BIDS

make\_dataset\_description(

path=bids\_path.root,

name=task,

authors=["Author1", "Author2"],

how\_to\_acknowledge=how\_to\_acknowledge,

acknowledgements="""\

Alexandre Gramfort, Mainak Jas, and Stefan Appelhoff prepared and updated the \

data in BIDS format.""",

data\_license="CC0",

ethics\_approvals=["placeholder"],

funding=[

"funding",

],

references\_and\_links=[

"https://doi.org/10.1016/j.neuroimage.2014.02.017",

"https://doi.org/10.3389/fnins.2013.00267",

"https://mne.tools/stable/documentation/datasets.html#sample",

],

doi="doi:10.18112/openneuro.ds000248.v1.2.4",

overwrite=True,

)

desc\_json\_path = bids\_path.root / "dataset\_description.json"

with open(desc\_json\_path, encoding="utf-8-sig") as fid:

pprint(json.loads(fid.read()))

# %%

# This should be very similar to the `ds000248 dataset\_description.json`\_!

**Script for generating an events.json file**

MNE BIDs tutorial online for extracting JSON information from raw MEG files

<https://mne.tools/mne-bids/dev/auto_examples/convert_mne_sample.html#sphx-glr-auto-examples-convert-mne-sample-py>

Please note, I originally ran this script after I have done BIDS. You will need the bids\_folder path to run this script but it “may” be possible to generate an events.json file without doing BIDS first. However additional modifications of the script may be required.

The following script only needs to be run once per participant, and generates an **events.json** file for our participant data. (example events.json file uploaded separately)

import json

import os

import mne

# Define event fields with descriptions for **events.json**

event\_fields = {

"onset": {"Description": "The time at which the event occurs, in seconds relative to the beginning of the recording."},

"duration": {"Description": "The duration of the event, in seconds."},

"trial\_type": {"Description": "The type or condition of the trial/event. Types of images: God, Supernatural, Humans, Inanimate Objects."},

"response\_time": {"Description": "Response time in seconds."},

"stim\_file": {"Description": "The filename of the corresponding stimulus files."},

"stim\_channel": {"Description": " STI101 is the summation channel. STI001 to STI008 are for stimulus presentation. STI009 to STI012 are for button responses."}, #

"value": {"Description": "The value associated with the event\_id, to be found in the event\_dictionary."},

"sample": {"Description": "The sample index at which the event occurs, relative to the beginning of the recording."},

"question\_type": {"Description": "Question types: extent of mind, extent of morality, extent of connection."},

"task": {"Description": "The task associated with the event: image, resting, naturalistic."}

}

# Write event fields to events.json in the bids\_folder

bids\_folder = "sub\_07\_bids\_folder" #change this according to where your bids are stored for each participant data

events\_json\_path = os.path.join(bids\_folder, "events.json")

with open(events\_json\_path, 'w') as f:

json.dump(event\_fields, f, indent=4)

print(f"events.json file has been generated and stored in {events\_json\_path}.")

**Reading events from individual channels**

Note: The individual stimulus trigger channels are labelled STI001 to STI012. The channel that consolidates the stimulus triggers (the summation channel) is labelled STI101.

STI001 to STI008 are for stimulus presentation. STI009 to STI012 are for button responses.

**Script for reading stimulus trigger channel STI001 (this can be changed)**

import mne

# Load the raw MEG data first

raw = mne.io.read\_raw\_fif('AZLLGK\_task\_run1\_raw.fif', preload=True)

# List all channel names in the dataset

print(raw.info['ch\_names'])

events = mne.find\_events(raw, stim\_channel= 'STI001', output='onset',

consecutive='increasing', min\_duration=0,

shortest\_event=1/raw.info['sfreq']

, mask=None, uint\_cast=False,

mask\_type='and', initial\_event=False,

verbose=True)

print("Events found:")

print(events)

raw.plot() # a pop up screen will appear; scroll down to the very end and you can see the trigger event from each stimulus channel. You can zoom in/out to see the triggers better

- - - - - - - - After you run print(events), you will see this output: - - - - - - - - - - - -

A screenshot of a computer program

Description automatically generated

The 3rd column is the 5 voltage pulse. If you only look at one channel at one time, then the 3rd column will give you the voltage pulse number (which is 5).

Useful tidbit: The first column is sampling number. The MEG machine infers time by dividing the sampling number by the sampling frequency.

E.g. Sample number 32270 divided by 1000 then you get time (roughly speaking)

If interested, can read up more on documentations for **mne.find\_events** on MNE website.

- - - - - - - - - - - - - - -

If you look at the channel STI101, then it will give you event ID in 3rd column (example screenshot below)L

raw = mne.io.read\_raw\_fif('AZLLGK\_task\_run1\_raw.fif', preload=True)

# List all channel names in the dataset

print(raw.info['ch\_names'])

events = mne.find\_events(raw, stim\_channel= 'STI101', output='onset',

consecutive='increasing', min\_duration=0,

shortest\_event=1/raw.info['sfreq']

, mask=None, uint\_cast=False,

mask\_type='and', initial\_event=False,

verbose=True)

print("Events found:")

print(events)

A screenshot of a computer

Description automatically generated

In the third column, you now have the value number of each event\_ID.

If in doubt, use the raw.plot() function to visualize each trigger event in each individual stimulus channel. “Extra” event IDs are usually the result of overlapping events (e.g. two button presses that overlap with one another) and not a big concern as long as the total number of your desired stimulus conditions are as expected.