**EazzZyLearn documentation**

Version 20211129

Preambule

EazzZyLearn is the code that detects deep sleep and time windows for cueing for memory reactivation in real-time. When reaching these time windows, it will trigger a cue automatically. All parameters for this online analysis and stimulation have to be **set in the file *parameters.py*** in the root (main) folder.

EazzZyLearn is written entirely in Python. It can be executed in any terminal or script editor such as Microsoft Visual Studio Code by **executing the file *main.py***.

EazzZyLearn is executed after memory encoding (new memory from a memory task) and will semi-automatically select cues to play back during sleep.

[**Preambule**](#_5u3j9e7aiez5) **1**

[**Manual parameters (changing for every session)**](#_ivvpgfslsu9n) **1**

[Recording session information](#_eb4hjv4rfk2i) 1

[Channel information](#_jiy1u54oyp1) 2

[**Constant parameters (should not change)**](#_6g0u08oen5rn) **4**

[Data import](#_ib2hd9mnsaio) 4

[Cue-related](#_xqcu6yepws51) 5

[General data handling](#_j9o41f57mvt) 5

[**Data output formats**](#_wfwwepivvenj) **6**

[**Code execution and interaction with it**](#_wk8cjkc7itwu) **6**

[**Fundamental structure of the code**](#_xiim7rh88s3b) **7**

[**Classes and methods**](#_xb56t9utepy9) **7**

# 

# Manual parameters (changing for every session)

## Recording session information

Subject- and session-specific information

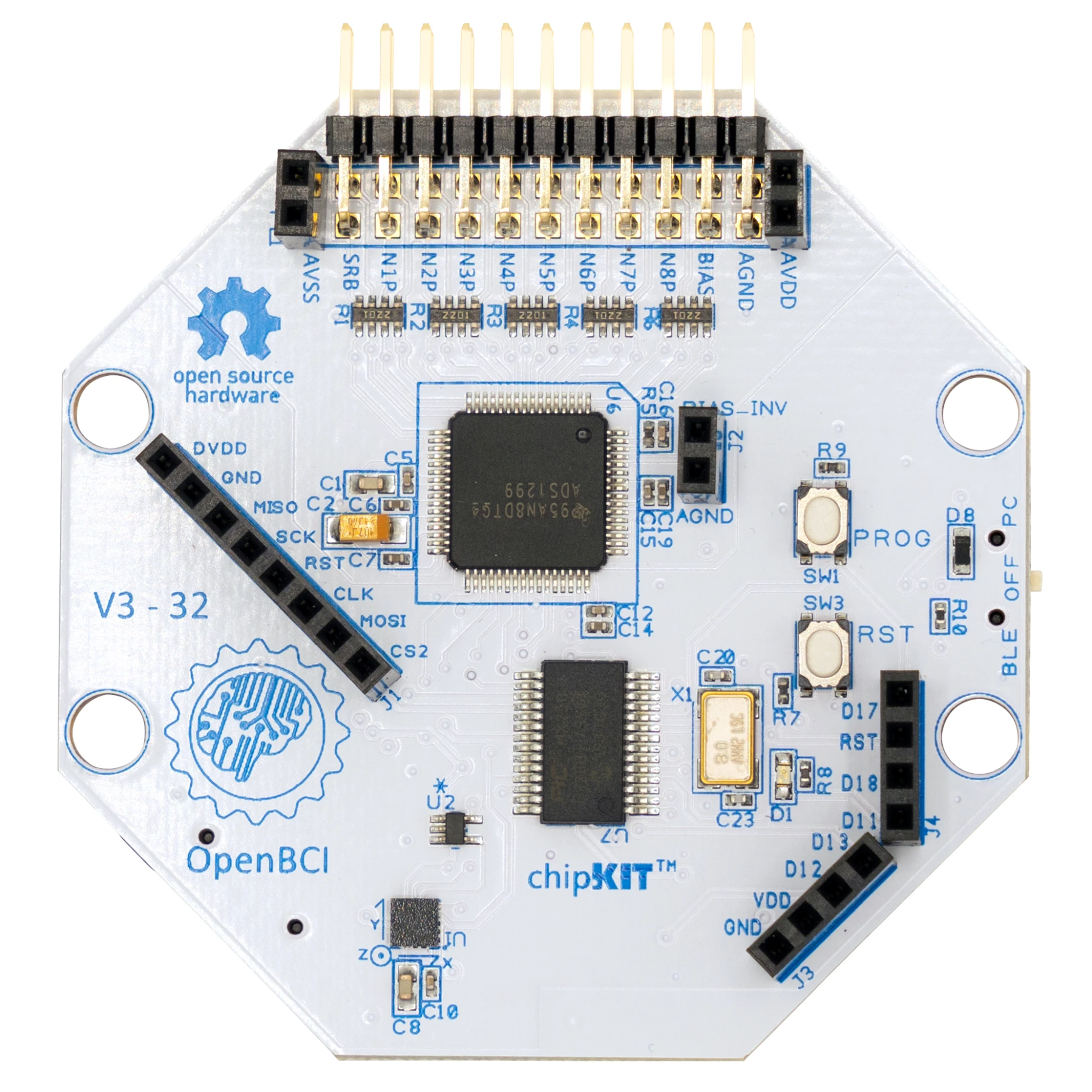
| OUTPUT\_DIR | Folder where all data outputs (stimulation information, EEG signals, …) will be stored |
| --- | --- |
| SUBJECT\_INFO | Important identifying information about the subject. Sample\_rate is constant = 250 Hz (OpenBCI) |
| CORRECT\_ANSWERS | Answers that were correct in the memory task game. Comment out (by placing a **“#”** in front of the line) the answers that were **NOT** correct |
| FORCE\_CUE\_LISTS | In case the stimulation code is restarted, you can predefine the list of stimulated cards with this parameter. Look at the output file header of when the code was run first time to see which cards had been chosen for stimulation |

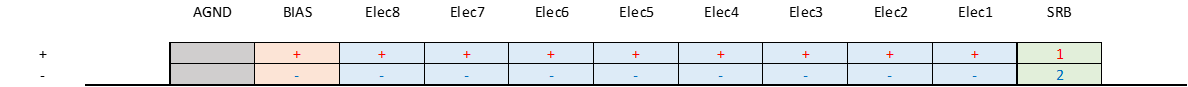
## Channel information

If necessary, you have to change the channel list based on where you placed scalp electrodes and where you plugged them in to OpenBCI [1]. Numbers in the list represent the pins occupied on the OpenBCI board. Keep in mind that **Python indices are starting with 0**. Therefore, 0 = 1, 1 = 2, … on the OpenBCI board!

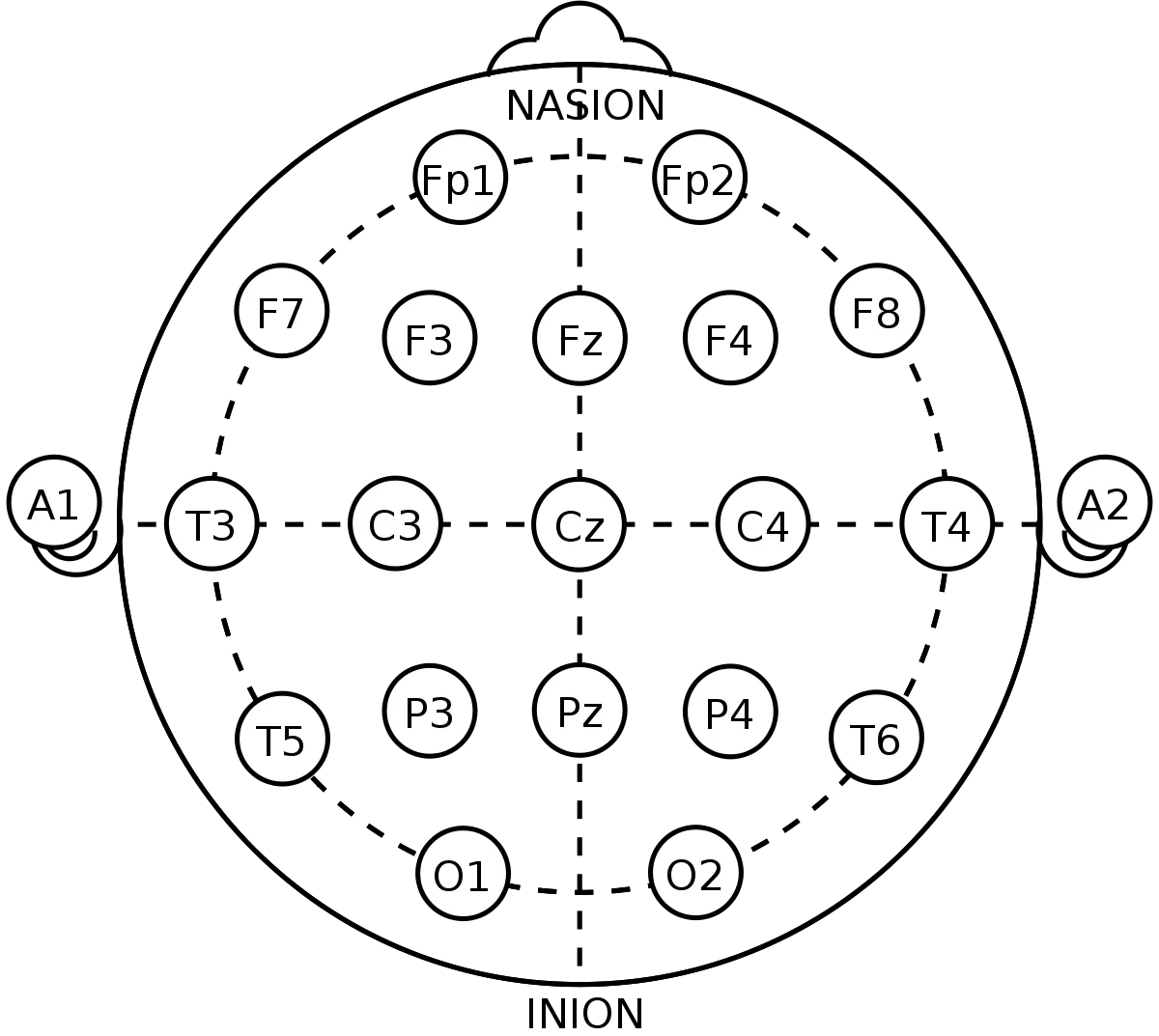
| NUM\_CHANNELS | Information of the data structure coming from OpenBCI. This indicates the total number of channels available on the board, **NOT** the number of channels used (pins occupied)! |
| --- | --- |
| elec | Python dictionary where [“XXX”] XXX is the name of the channel and the number = xxx the index of the channel on the board (careful with Python indexing starting at 0!) |
| IDX\_ELEC | The index of the channel that will be used by default. Slow oscillations are best detected in the frontal area [2]. The index has to be set by **defining the dictionary entry** Dictionary[“Key”] (ie elec["Fp2"]) |

*[1] Illustration of the OpenBCI Cython board and pin organization*

**

**

*[2] Illustration of the 10-20 scalp EEG system*



# Constant parameters (should not change)

## Data import

| IP | IP address of the data transfer protocol: The data from OpenBCI is imported by the code over UDP socket communication. Has to match the one in OpenBCI GUI > Networking > UDP > IP |
| --- | --- |
| PORT | Same as IP above, but for the port. Has to match the one in OpenBCI GUI > Networking > UDP > Port |
| MAIN\_BUFFER\_LENGTH | Buffer length in milliseconds (ms) of the imported data. The longer the buffer, the **more accurate** the code **but the slower** the code as well. |
| DELTA\_BUFFER\_LENGTH | Length (ms) of the time window that will be used to extract slow oscillations. Arbitrary length, but has to be long enough to contain at least one slow oscillation (up to 2000 ms per slow oscillation) |
| THRESHOLD\_BUFFER\_LENGTH | Length (ms) of the time window that will be used to determine the minimum downstate amplitude for a slow oscillation to have in order to be considered valid for stimulation. [**3 for explanation**] |
| SLEEP\_BUFFER\_LENGTH | The vector length (ms) of the signal used for sleep staging. **30s time windows** are standard in sleep research. |
| REPONSE\_BUFFER\_LENGTH | Length (ms) of time window used for detecting awakening of the subject. The shorter the window the faster we detect wakening but the more the signal fluctuations (inaccuracy) |

[3] After every stimulation, we have to expect that the brain gets less synchronized because of the sound “perturbation” (amplitudes of slow oscillation downstates and upstates decreasing) and might even shift to lighter sleep stages. Here we assure that we only consider downstates of slow oscillations that are at least as synchronized as the ones inside the time window (length defined by threshold vector length) before. The longer the more aggressive the threshold is and the less we will stimulate. The shorter the more we stimulate, but the more we risk for subjects to slowly wake up.

## Cue-related

| LEN\_REFRACTORY | Number (seconds) which defines how long after a cue stimulation, we “leave the brain in peace” without triggering any subsequent stimulation even if there is a slow oscillation (indication from Schreiner et al., 2015, Nature Communications that cue+feedback destroys CMR effects) |
| --- | --- |
| DEFAULT\_THRESHOLD | Default negative amplitude (microvolts) under which a slow oscillation throw is considered valid |
| NON\_PHYSIOLOGICAL\_THRESHOLD | This inversibly of the DEFAULT\_THRESHOLD invalidates negative amplitudes that are too low as large fluctuations from muscle movements could be interpreted as slow oscillation throws |
| SLEEP\_STAGE\_INTERVAL | How often (seconds) the code is evaluating the sleep stage of the subject (differentiates between Slow Wave Sleep (SWS) and any “other” stage) |
| WAKE\_STAGE\_INTERVAL | How often (seconds) the code is evaluating the wake stage of the subject (differentiates between is or is not awake) |
| SLEEP\_THRESHOLDS | Values (unitless or PSD values) that help wake and sleep staging processes to distinguish between stages. Values are based on observations |
| WAKE\_THRESHOLDS | Same as SLEEP\_THRESHOLDS but for wake staging. |
| FREQUENCY\_BANDS | Frequency limits for general filtering purposes and power estimations of frequency bands |
| FILT\_ORDER | Order to the butter window used for filtfilt method |

## General data handling

| DATA\_SAVE\_INTERVAL | How often the data from OpenBCI gets written to disk |
| --- | --- |
| ENCODING | Specifying data output format for easier import later |

# Data output formats

All data outputs are stored in plain text files .txt. All files contain the same header information about the subject/recording/parameters used. Data is stored in a comma separated manner in columns where the first is always the time stamp of the occurrence of the data in milliseconds.

Three different outputs are generated:

1. “[...]**\_eeg**.txt”: contains the raw signal from all channels (even non-used ones) where columns are channels and rows are signal samples at different time stamps
2. “[...]**\_stages**.txt”: contains all information about the current wake and sleep stage evaluations of the subject
3. “[...]**\_pred**.txt”: contains time stamps of detected downstates and predicted upstates
4. “[...]**\_stim**.txt”: Contains several information:

* Stimulations and which cue has been presented
* Softstate of the code (that is, we can manually block/force stimulations)
* Switches between electrode channels for slow oscillation prediction and stimulation
* Lastly, all predicted slow oscillation upstates

An anticipated **ending/quitting of the code will not lead to any data loss if executed correctly** (specified in section “Interaction with code”).

# Code execution and interaction with it

The code is executed once (for example python -m ./main.py) and will then run until stopped manually. During execution, several options are available by key presses:



Numeric **keys 1 to 8 will switch** the slow oscillation prediction and stimulation to using the electrode X where X is the pressed key.

Pressing **P** will pause the stimulation of any detected slow oscillation. During this pause, raw signal storage as well as sleep/wake staging is still ongoing.

Pressing **R** (default stage of the code) will resume the stimulation upon slow oscillation detection normally.

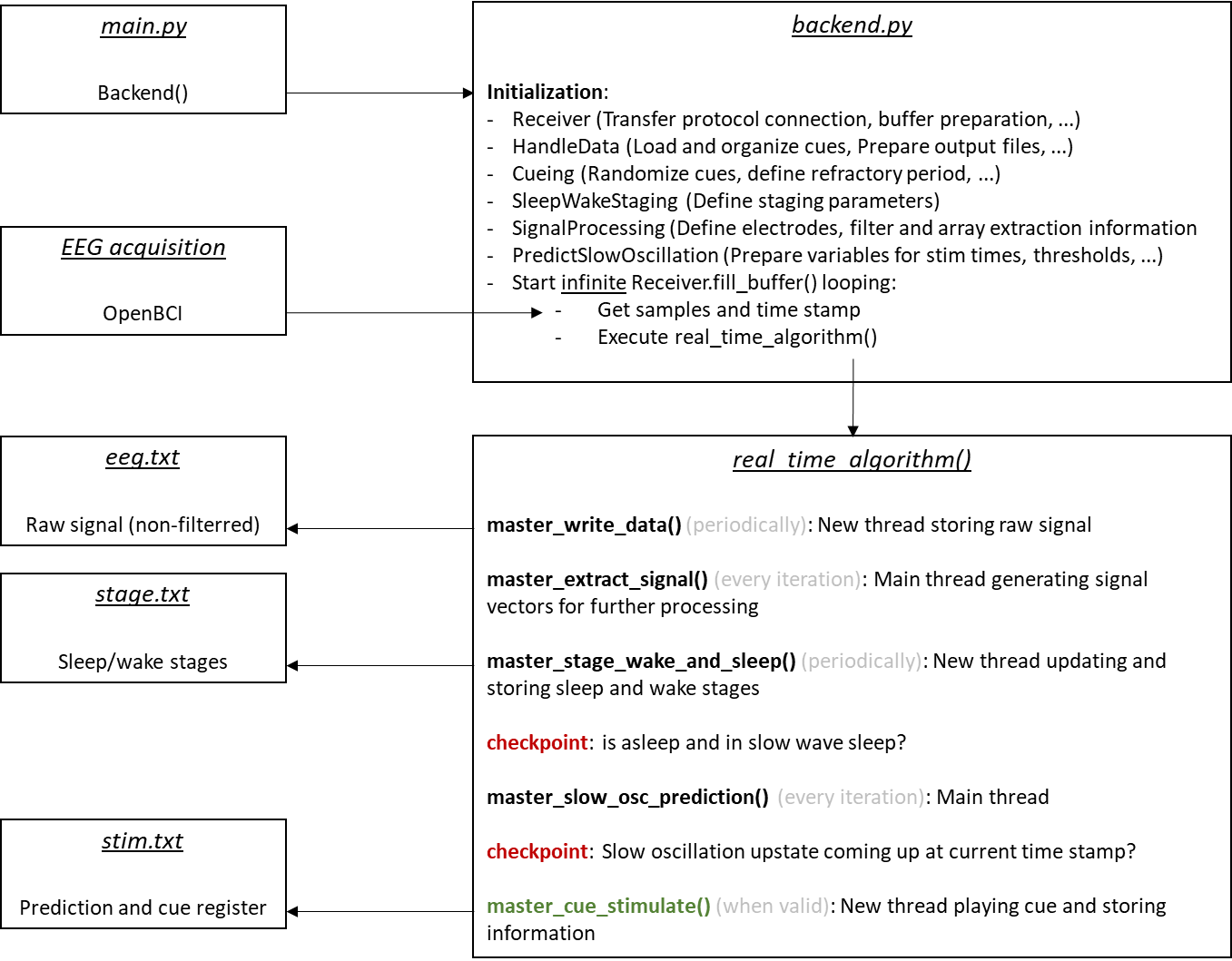
Pressing **F** will ignore sleep/wake staging evaluations and force the stimulation of slow oscillations.

Pressing **Q** will quit the program. **This is the only correct way of ending the recording**! You are requested to answer with “Y” in order to quit the program definitely or any other answer if you accidently hit Q and want to continue recording.

# 

# Fundamental structure of the code

General working scheme of the code shown below:



# Classes and methods

The code contains several classes (collection of methods) that are responsible for different aspects of the code:

1. **Backend**: Handles all the initialization on the classes themselves and contains the real\_time\_algorithm() method that controls everything.

* \_\_init\_\_: Initializes all the Classes’ \_\_init\_\_ and starts the receiver

1. **Receiver**: In charge of importing the EEG signals into the program and to call the real\_time\_algorithm() infinitely as well as to define the softstate of stimulations.

* prep\_socket
* prep\_buffer
* prep\_time\_stamps
* get\_sample
* get\_time\_stamp
* fill\_buffer
* start\_receiver
* stop\_receiver
* define\_stimulation\_state

1. **HandleData**: In charge of all the parameters for importing and exporting of information

* prep\_output\_dir
* format\_subject\_name
* prep\_files
* master\_write\_data
* write\_data\_thread
* prep\_cue\_dir
* prep\_cue\_list
* prep\_cue\_selec
* prep\_cue\_info
* prep\_cue\_load

1. **SignalProcessing**: Takes care of the extraction of physiological scalp EEG signals (frequencies between 0.1 and 45 Hz, and notch filtered) from the set electrode channel.

* filt\_signal
* extract\_array
* master\_extract\_signal
* switch\_channel

1. **SleepWakeState**: In charge of the evaluation of wakings of the subjects as well as their sleep stages.

* master\_stage\_wake\_and\_sleep
* stage\_wake\_thread
* stage\_sleep\_thread
* staging
* band\_ratio\_thresholding
* stage\_write
* power\_spectr\_multitaper
* power\_spectr\_welch

1. **PredictSlowOscillation**: Detects slow oscillation downstates and predicts upstates

* set\_threshold
* extract\_slow\_oscillation\_onset
* downstate\_validation
* throw\_validation
* downstate\_mirroring
* sine\_wave\_fitting
* correct\_stim\_time
* master\_slow\_osc\_prediction

1. **Cueing**: Handles cue stimulation of the subjects

* cue\_randomize
* cue\_play
* cue\_write
* master\_cue\_stimulate