BrainFlow Documentation

Andrey Parfenov

Jun 05, 2021

CONTENTS

BrainFlow is a library intended to obtain, parse and analyze EEG, EMG, ECG and other kinds of data from biosensors.

It provides a **uniform data acquisition API for all supported boards**, it means that you can switch boards without any changes in code and applications on top of BrainFlow are board agnostic. Also there is **powerful API to perform signal processing** which you can use even without BCI headset. Both of these two APIs are the same across bindings.

CHAPTER

ONE

SUPPORTED BOARDS

To create an instance of BoardShim class for your board check required inputs in the table below:

Table 1: Required inputs

Board	Board Id	BrainFlowInputParams.serial_port	BrainFlowInputParams.mac_address	BrainFlowInputParams.ip_address	BrainFlowInputParams.ip_port	BrainFlowInputParams.ip_protocol	BrainFlowInputParams.other_info	BrainFlowInputParams.timeout	BrainFlowInputParams.serial_number	BrainFlowInputParams.file
Playback Board	BoardIds.PLAYBACK_FILE_BOARD (-3)	•	•	•	•	•	Board Id of master board	•	•	path to file for playback
Streaming Board	BoardIds.STREAMING_BOARD (-2)	•	•	multicast IP address	port	•	Board Id of master board	•	•	•
Synthetic Board	BoardIds.SYNTHETIC_BOARD (-1)	•	•	•	•	•	•	•	•	•
Cyton	BoardIds.CYTON_BOARD (0)	dongle serial port(COM3, /dev/ttyUSB0, /dev/cu.usbserial-xxxxxx…)	•	•	•	•	•	•	•	•
Ganglion	BoardIds.GANGLION_BOARD (1)	dongle serial port(COM3, /dev/ttyUSB0…)	Optional: Ganglion's MAC address	•	•	•	•	Timeout for device discovery(default 15sec)	•	•
Cyton Daisy	BoardIds.CYTON_DAISY_BOARD (2)	dongle serial port(COM3, /dev/ttyUSB0, /dev/cu.usbserial-xxxxxx…)	•	•	•	•	•	•	•	•
Ganglion WIFI	BoardIds.GANGLION_WIFI_BOARD (4)	•	•	WIFI Shield IP(default 192.168.4.1)	any local port which is free	•	•	Timeout for HTTP response(default 10sec)	•	•
Cyton WIFI	BoardIds.CYTON_WIFI_BOARD (5)	•	•	WIFI Shield IP(default 192.168.4.1)	any local port which is free	•	•	Timeout for HTTP response(default 10sec)	•	•
Cyton Daisy WIFI	BoardIds.CYTON_DAISY_WIFI_BOARD (6)	•	•	WIFI Shield IP(default 192.168.4.1)	any local port which is free	•	•	Timeout for HTTP response(default 10sec)	•	•
BrainBit	BoardIds.BRAINBIT_BOARD (7)	•	•	•	•	•	•	Timeout for device discovery(default 15sec)	Optional: Serial Number of BrainBit device	•
Unicorn	BoardIds.UNICORN_BOARD (8)	•	•	•	•	•	•	•	Optional: Serial Num-	•

1.1 Playback File Board

This board playbacks file recorded using another BrainFlow board.

It allows you to test signal processing algorithms on real data without device.

To choose this board in BoardShim constructor please specify:

- board_id: -3
- other_info field of BrainFlowInputParams structure should contain board_id of device used to create playback file
- file field of BrainFlowInputParams structure

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS

By default it generates new timestamps and stops at the end of the file. You can override it using commands:

```
board.config_board ('loopback_true')
board.config_board ('loopback_false')
board.config_board ('new_timestamps')
board.config_board ('old_timestamps')
```

In methods like:

```
get_eeg_channels (board_id)
get_emg_channels (board_id)
get_ecg_channels (board_id)
# .......
```

You need to use master board id instead Playback Board Id, because exact data format for playback board is controlled by master board as well as sampling rate.

Board Specs:

- num eeg(emg,...) channels: like in master board
- num acceleration channels: like in master board
- sampling rate: like in master board
- communication: None

1.2 Streaming Board

BrainFlow's boards can stream data to different destinations like file, socket and so on. This board acts like a consumer for data streamed from the main process.

To use it in the first process you should call:

```
# choose any valid multicast address(from "224.0.0.0" to "239.255.255.255") and port
start_stream (450000, 'streaming_board://225.1.1.1:6677')
```

In the second process please specify:

- board_id: -2
- ip_address field of BrainFlowInputParams structure, for example above it's 225.1.1.1
- ip_port field of BrainFlowInputParams structure, for example above it's 6677
- other_info field of BrainFlowInputParams structure, write there board_id for a board which acts like data provider(master board)

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS

In methods like:

```
get_eeg_channels (board_id)
get_emg_channels (board_id)
get_ecg_channels (board_id)
# .......
```

You need to use master board id instead Streaming Board Id, because exact data format for streaming board is controlled by master board as well as sampling rate.

Board Specs:

- num eeg(emg,...) channels: like in master board
- num acceleration channels: like in master board
- sampling rate: like in master board
- communication: UDP multicast socket to read data from master board

1.3 Synthetic Board

This board generates synthetic data and you dont need real hardware to use it.

It can be extremely useful during development.

To choose this board in BoardShim constructor please specify:

- board_id: -1
- you dont need to set any fields in BrainFlowInputParams structure

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Android

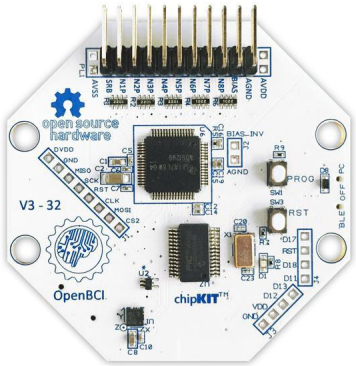
Board Specs:

- num eeg(emg,...) channels: 8
- num acceleration channels: 3
- sampling rate: 256

- communication: None

1.4 OpenBCI

1.4.1 Cyton



Cyton Getting Started Guide from OpenBCI

To choose this board in BoardShim constructor please specify:

- board_id: 0
- serial_port field of BrainFlowInputParams structure

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS

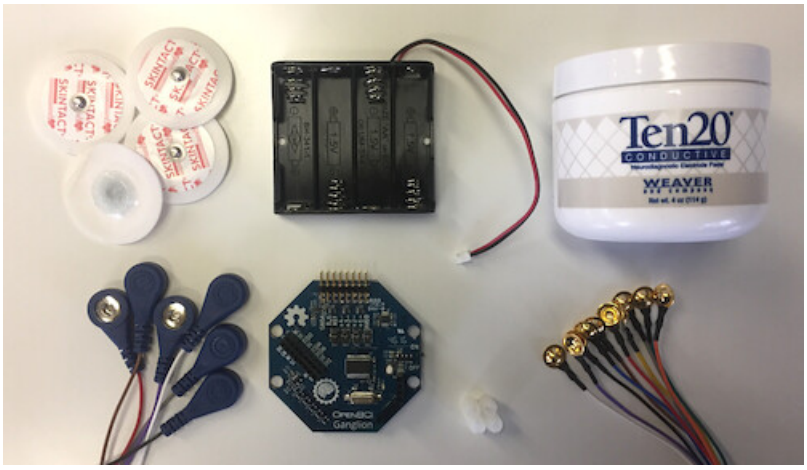
On MacOS there are two serial ports for each device: /dev/tty.... and /dev/cu.... You HAVE to specify /dev/cu....

Also, on Unix-like systems you may need to configure permissions for serial port or run with sudo.

Board Spec:

- num eeg(emg,...) channels: 8
- num acceleration channels: 3
- sampling rate: 250
- communication: serial port
- signal gain: 24

1.4.2 Ganglion



Ganglion Getting Started Guide from OpenBCI

To use Ganglion board you need a dongle

Also, on Unix-like systems you may need to configure permissions for serial port or run with sudo.

To choose this board in BoardShim constructor please specify:

- board_id: 1
- serial_port field of BrainFlowInputParams structure
- mac_address field of BrainFlowInputParams structure, if its empty BrainFlow will try to autodiscover Ganglion
- optional: timeout field of BrainFlowInputParams structure, default is 15sec

To get Ganglion's MAC address you can use:

- Windows: Bluetooth LE Explorer App
- Linux: hcitool command

Supported platforms:

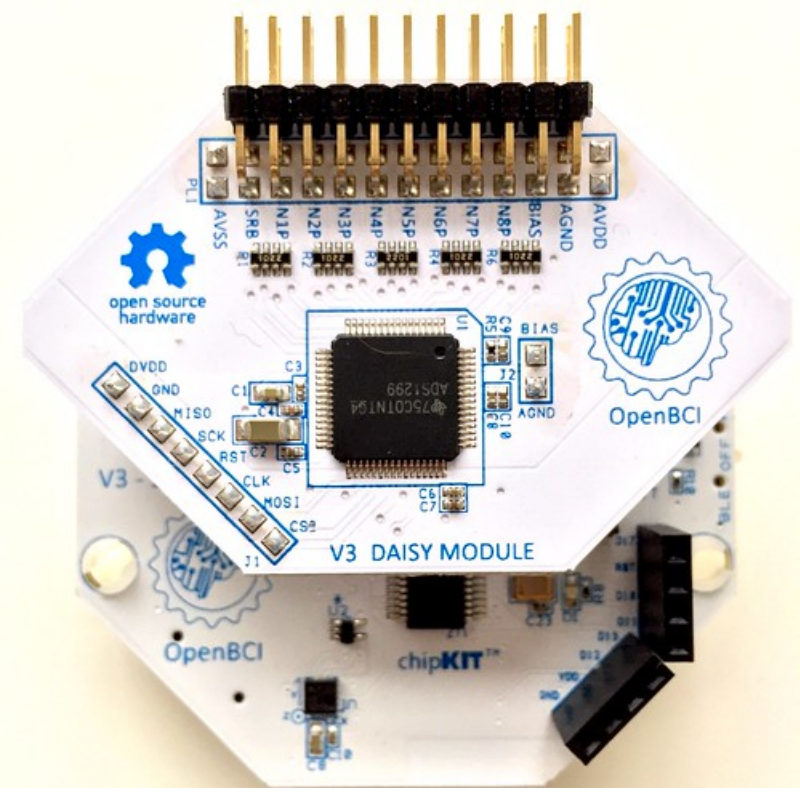
- Windows >= 8.1
- Linux
- MacOS

On MacOS there are two serial ports for each device: /dev/tty.….. and /dev/cu.….. You HAVE to specify /dev/cu.…..

Board Spec:

- num eeg(emg,…) channels: 4
- num acceleration channels: 3
- sampling rate: 200
- communication: Bluetooth Low Energy behind serial port from the dongle

1.4.3 Cyton Daisy



CytonDaisy Getting Started Guide from OpenBCI

To choose this board in BoardShim constructor please specify:

- board_id: 2
- serial_port field of BrainFlowInputParams structure

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS

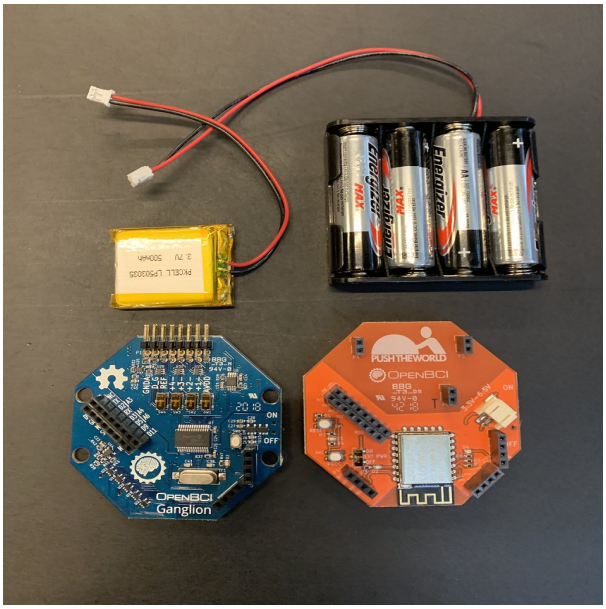
On MacOS there are two serial ports for each device: /dev/tty..... and /dev/cu..... You HAVE to specify /dev/cu.....

Also, on Unix-like systems you may need to configure permissions for serial port or run with sudo.

Board Spec:

- num eeg(emg,...) channels: 16
- num acceleration channels: 3
- sampling rate: 125
- communication: serial port
- signal gain: 24

1.4.4 Ganglion with WIFI Shield



WIFI Shield Getting Started Guide from OpenBCI

WIFI Shield Programming Guide from OpenBCI

To choose this board in BoardShim constructor please specify:

- board_id: 4
- ip_address field of BrainFlowInputParams structure should contain WiFi Shield Ip address(in direct mode its 192.168.4.1), if it's empty BrainFlow will try to autodiscover WIFI Shield and in case of failure will try to use 192.168.4.1
- ip_port field of BrainFlowInputParams structure should be any local port which is free right now
- optional: timeout field of BrainFlowInputParams structure, default is 10sec

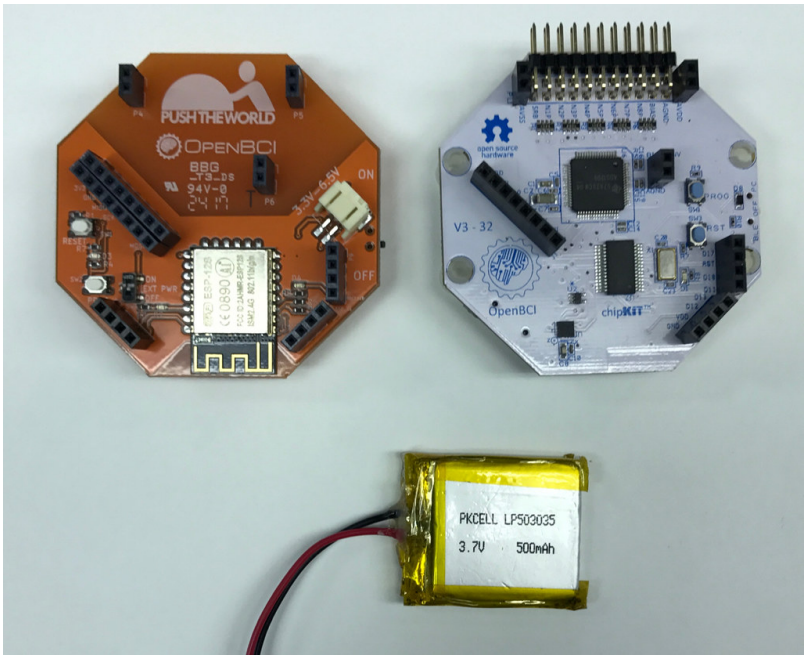
Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Android

Board Spec:

- num eeg(emg,...) channels: 4
- num acceleration channels: 3
- sampling rate: 1600
- communication: TCP socket to read data and HTTP to send commands

1.4.5 Cyton with WIFI Shield



WIFI shield Getting Started Guide from OpenBCI

WIFI shield Programming Guide from OpenBCI

To choose this board in BoardShim constructor please specify:

- board_id: 5
- ip_address field of BrainFlowInputParams structure should contain WiFi Shield Ip address(in direct mode its 192.168.4.1), if it's empty BrainFlow will try to autodiscover WIFI Shield and in case of failure will try to use 192.168.4.1
- ip_port field of BrainFlowInputParams structure should be any local port which is free right now
- optional: timeout field of BrainFlowInputParams structure, default is 10sec

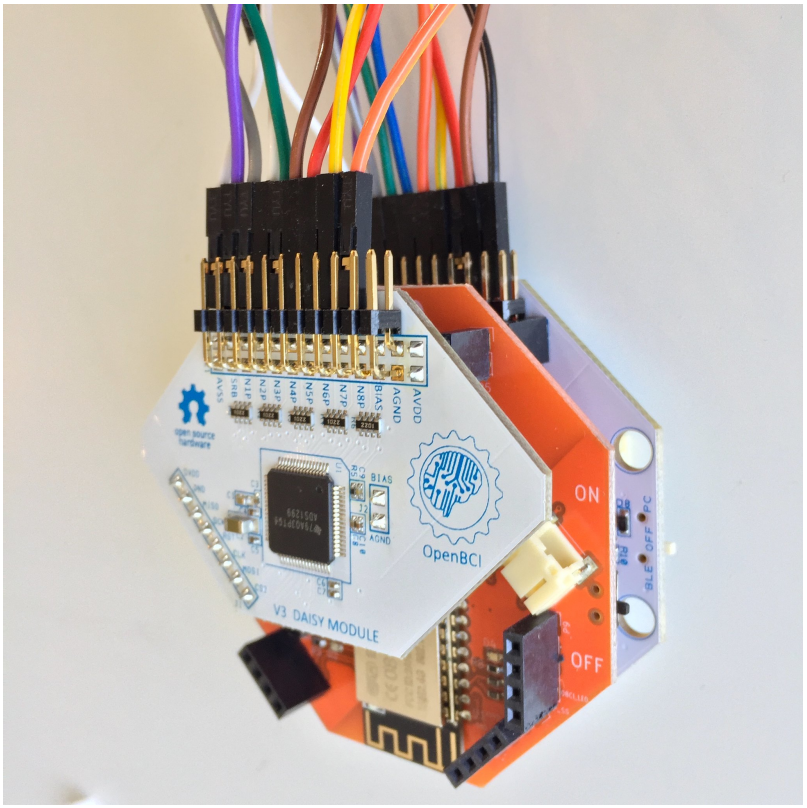
Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Android

Board Spec:

- num eeg(emg,. . .) channels: 8
- num acceleration channels: 3
- sampling rate: 1000
- communication: TCP socket to read data and HTTP to send commands
- signal gain: 24

1.4.6 CytonDaisy with WIFI Shield



WIFI Shield Getting Started Guide from OpenBCI

WIFI Shield Programming Guide from OpenBCI

To choose this board in BoardShim constructor please specify:

- board_id: 6
- ip_address field of BrainFlowInputParams structure should contain WiFi Shield Ip address(in direct mode its 192.168.4.1), if it's empty BrainFlow will try to autodiscover WIFI Shield and in case of failure will try to use 192.168.4.1
- ip_port field of BrainFlowInputParams structure should be any local port which is free right now
- optional: timeout field of BrainFlowInputParams structure, default is 10sec

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Android

Board Spec:

- num eeg(emg,...) channels: 16
- num acceleration channels: 3
- sampling rate: 1000

- communication: TCP socket to read data and HTTP to send commands
- signal gain: 24

1.5 NeuroMD

1.5.1 BrainBit

BrainBit website

To choose this board in BoardShim constructor please specify:

- board_id: 7
- optional: serial_number field of BrainFlowInputParams structure should contain Serial Number of BrainBit device, use it if you have multiple devices
- optional: timeout field of BrainFlowInputParams structure, default is 15sec

Supported platforms:

- Windows >= 10
- MacOS

Board Spec:

- num eeg channels: 4
- num acceleration channels: None
- sampling rate: 250

- communication: Bluetooth Low Energy

1.5.2 BrainBitBLED

This board allows you to use BLED112 dongle instead native API to work with BLE. Unlike original BrainBit libraries it works on Linux and devices like Raspberry Pi.

To choose this board in BoardShim constructor please specify:

- board_id: 18
- serial port field of BrainFlowInputParams structure
- optional: MAC address for your BrainBit device

Supported platforms:

- Windows
- MacOS
- Linux
- Devices like Raspberry Pi

Board Spec:

- num eeg channels: 4
- num acceleration channels: None
- sampling rate: 250
- communication: Bluetooth Low Energy with serial port dongle

1.5.3 Callibri(Yellow)

Callibri website

Callibri can be used to record EMG, ECG and EEG, but based on signal type you need to apply different settings for device.

BrainFlow does it for you, so there are:

- CALLIBRI_EEG_BOARD (board_id 9)
- CALLIBRI_EMG_BOARD (board_id 10)
- CALLIBRI_ECG_BOARD (board_id 11)

To choose this board in BoardShim constructor please specify:

- board_id: 9, 10 or 11 based on data type
- optional: to use electrodes connected vis USB write "ExternalSwitchInputMioUSB" to other_info field of BrainFlowInputParams structure
- optional: timeout field of BrainFlowInputParams structure, default is 15sec

Supported platforms:

- Windows >= 10
- MacOS

Board Spec:

- num exg channels: 1
- num acceleration channels: None

- communication: Bluetooth Low Energy

1.6 G.TEC

1.6.1 Unicorn

Unicorn website

To choose this board in BoardShim constructor please specify:

- board_id: 8
- optional: serial_number field of BrainFlowInputParams structure should contain Serial Number of BrainBit device, use it if you have multiple devices

Supported platforms:

- Ubuntu 18.04, may work on other Linux OSes, it depends on dynamic library provided by Unicorn
- Windows
- May also work on Raspberry PI, if you replace libunicorn.so by library provided by Unicorn for Raspberry PI

Steps to Setup:

- Connect the dongle
- Make sure that you paired Unicorn device with PC using provided dongle instead built-in Bluetooth

Board Spec:

- num eeg channels: 8
- num acceleration channels: 3
- sampling rate: 250
- communication: Bluetooth Low Energy

1.7 Neurosity

1.7.1 Notion 1

Notion website

Link to Neurosity Tutorial

To choose this board in BoardShim constructor please specify:

- board_id: 13
- optional: Serial Number field of BrainFlowInputParams structure, important if you have multiple devices in the same place

Supported platforms:

- Windows
- Linux
- MacOS

Note: On Windows you may need to disable firewall to allow broadcast messages.

Board Spec:

- num eeg channels: 8
- sampling rate: 250
- communication: UDP BroadCast

1.7.2 Notion 2

Notion website

Link to Neurosity Tutorial

To choose this board in BoardShim constructor please specify:

- board_id: 14
- optional: Serial Number field of BrainFlowInputParams structure, important if you have multiple devices in the same place

Supported platforms:

- Windows
- Linux
- MacOS

Note: On Windows you may need to disable firewall to allow broadcast messages.

Board Spec:

- num eeg channels: 8
- sampling rate: 250
- communication: UDP BroadCast

1.8 OYMotion

1.8.1 gForcePro ArmBand

OYMotion website

To choose this board in BoardShim constructor please specify:

- board_id: 16

Supported platforms:

- Windows

Note: Unlike other boards it returns ADC value instead uV.

Board Spec:

- num emg channels: 8

- sampling rate: 500

1.8.2 gForceDual ArmBand

OYMotion website

To choose this board in BoardShim constructor please specify:

- board_id: 19

Supported platforms:

- Windows

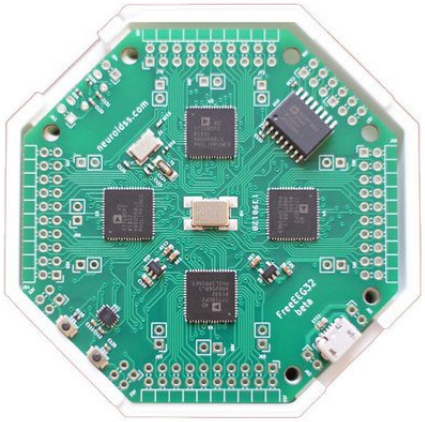
Note: Unlike other boards it returns ADC value instead uV.

Board Spec:

- num emg channels: 2
- sampling rate: 500

1.9 FreeEEG32

1.9.1 FreeEEG32



CrowdSupply

To choose this board in BoardShim constructor please specify:

- board_id: 17
- serial_port field of BrainFlowInputParams structure

On Unix-like systems you may need to configure permissions for serial port or run with sudo.

Supported platforms:

- Windows
- Linux
- MacOS

Board Spec:

- num eeg channels: 32
- sampling rate: 512
- communication: Serial Port

CHAPTER

TWO

INSTALLATION INSTRUCTIONS

2.1 Python

Please, make sure to use Python 3+. Next, install the latest release from PYPI with the following command in terminal

```
python -m pip install brainflow
```

If you want to install it from source files or build unreleased version from Github, you should compile core module first and run

```
cd python-package
python -m pip install -U .
```

2.2 C#

Windows(Visual Studio)

You are able to install the latest release from Nuget or build it yourself:

- Compile BrainFlow's core module
- open Visual Studio Solution
- install required nuget packages
- build it using Visual Studio
- **make sure that unmanaged(C++) libraries exist in search path** - set PATH env variable or copy them to correct folder

Unix(Mono)

- Compile BrainFlow's core module
- install nuget and Mono on your system
- install required nuget packages
- build it using Mono
- **make sure that unmanaged(C++) libraries exist in search path** - set LD_LIBRARY_PATH env variable or copy them to correct folder

Example for Fedora:

```
# compile c++ code
tools/build_linux.sh
# install dependencies, we skip dnf configuration steps
sudo dnf install nuget
sudo dnf install mono-devel
sudo dnf install mono-complete
sudo dnf install monodevelop
# install nuget packages
nuget restore csharp-package/brainflow/brainflow.sln
# build solution
xbuild csharp-package/brainflow/brainflow.sln
# run tests
export LD_LIBRARY_PATH=/home/andreyparfenov/brainflow/installed_linux/lib/
mono csharp-package/brainflow/denoising/bin/Debug/test.exe
```

2.3 R

R binding is based on reticulate package and calls Python code, so you need to install Python binding first, make sure that reticulate uses correct virtual environment, after that you will be able to build R package from command line or using R Studio, install it and run samples.

2.4 Java

You are able to download jar files directly from release page

If you want to install it from source files or build unreleased version from github you should compile core module first and run

```
cd java-package
cd brainflow
mvn package
```

Also, you can use GitHub Package and download BrainFlow using Maven or Gradle. To use Github packages you need to change Maven settings. Example file here you need to change OWNER and TOKEN by Github username and token with an access to Github Packages.

2.5 Matlab

Steps to setup Matlab binding for BrainFlow:

- Compile Core Module, using instructions below. If you don't want to compile C++ code you can download Matlab package with precompiled libs from Release page
- Open Matlab IDE and open brainflow/matlab-package/brainflow folder there
- Add folders lib and inc to Matlab path
- If you want to run Matlab scripts from folders different than brainflow/matlab-package/brainflow you need to add it to your Matlab path too

2.6 Julia

BrainFlow is a registered package in the Julia general registry, so it can be installed via the Pkg manager:

Example:

```
import Pkg
Pkg.add("BrainFlow")
```

When using BrainFlow for the first time in Julia, the BrainFlow artifact containing the compiled BrainFlow libraries will be downloaded from release page automatically.

If you compile BrainFlow from source local libraries will take precedence over the artifact.

2.7 Docker Image

There are docker images with precompiled BrainFlow. You can get them from DockerHub.

All bindings except Matlab are preinstalled there and libraries compiled with OpenMP support.

Also, there are other packages for BCI research and development:

- mne
- pyriemann
- scipy
- matplotlib
- jupyter
- pandas
- etc

If your devices uses TCPIP to send data, you need to run docker container with `--network host`. For serial port connection you need to pass serial port to docker using `--device %your port here%`

Example:

```
# pull container from DockerHub
docker pull brainflow/brainflow:3.7.2
# run docker container with serial port /dev/ttyUSB0
docker run -it --device /dev/ttyUSB0 brainflow/brainflow:3.7.2 /bin/bash
# run docker container for boards which use networking
docker run -it --network host brainflow/brainflow:3.7.2 /bin/bash
```

2.8 Compilation of Core Module and C++ Binding

2.8.1 Windows

- Install CMake>=3.13 you can install it from PYPI via pip
- Install Visual Studio 2017, you can use another version but you will need to change CMake generator in batch files or run CMake commands manually. Also in CI we test only VS2017
- In VS installer make sure you selected “Visual C++ ATL support”

- Build it as a CMake project manually or use cmd files from tools directory

Compilation using cmd files:

```
python -m pip install cmake
# need to run these files from project dir
.\tools\build_win32.cmd
.\tools\build_win64.cmd
```

2.8.2 Linux

- Install CMake>=3.13 you can install it from PYPI via pip
- If you are going to distribute compiled Linux libraries you HAVE to build it inside manylinux Docker container
- Build it as a CMake project manually or use bash file from tools directory
- You can use any compiler but for Linux we test only GCC, also we test only 64bit libraries for Linux

Compilation using bash file:

```
python -m pip install cmake
# you may need to change line endings using dos2unix or text editor for file below
# need to run this file from project dir
bash ./tools/build_linux.sh
```

2.8.3 MacOS

- Install CMake>=3.13 you can install it from PYPI via pip
- Build it as a CMake project manually or use bash file from tools directory
- You can use any compiler but for MacOS we test only Clang

Compilation using bash file:

```
python -m pip install cmake
# you may need to change line endings using dos2unix or text editor for file below
# need to run this file from project dir
bash ./tools/build_mac.sh
```

2.8.4 Compilation with OpenMP

Some data processing and machine learning algorithms work much faster if you run them in multiple threads. To parallel computations we use OpenMP library.

Precompiled libraries which you download from PYPI/Nuget/Maven/etc built without OpenMP support and work in single thread.

If you want to increase performance of signal processing algorithms you can compile BrainFlow from the source and turn on *USE_OPENMP* option.

To build BrainFlow with OpenMP support first of all you need to install OpenMP.

- On Windows all you need is Visual C++ Redist package which is installed automatically with Visual Studio
- On Linux you may need to install libgomp if it's not currently installed

- On MacOS you need to run `brew install libomp`

After that you need to compile BrainFlow with OpenMP support, steps are exactly the same as above, but you need to run bash or cmd scripts whith _omp postfix.

Example:

```
# for Linux
bash ./tools/build_linux_omp.sh
# for MacOS
bash ./tools/build_mac_omp.sh
# for Windows
.\tools\build_win64_omp.cmd
```

If you use CMake directly to build BrainFlow you need to add `-DUSE_OPENMP=ON` to CMake config command line.

2.9 Android

To check supported boards for Android visit *Supported Boards*

2.9.1 Installation instructions

- Create Java project in Android Studio, Kotlin is not supported
- Download *jniLibs.zip* from Release page
- Unpack *jniLibs.zip* and copy it's content to *project/app/src/main/jniLibs*
- Download *brainflow-jar-with-dependencies.jar* from Release page or from Github package
- Copy *brainflow-jar-with-dependencies.jar* to *project/app/libs folder*

Now you can use BrainFlow SDK in your Android application!

Note: Android Studio inline compiler may show red errors but it should be compiled fine with Gradle. To fix inline compiler you can use *File > Sync Project with Gradle Files* or click at *File > Invalidate Cache/Restart > Invalidate and Restart*

For some API calls you need to provide additional permissions via manifest file of your application

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE"></uses-
↪permission>
<uses-permission android:name="android.permission.READ_EXTERNAL_STORAGE"></uses-
↪permission>
<uses-permission android:name="android.permission.WRITE_EXTERNAL_STORAGE"></uses-
↪permission>
```

2.9.2 Compilation using Android NDK

For BrainFlow developers

To test your changes in BrainFlow on Android you need to build it using Android NDK manually.

Compilation instructions:

- Download Android NDK
- Download Ninja or get one from the *tools* folder, make sure that *ninja.exe* is in search path
- You can also try *MinGW Makefiles* instead *Ninja*, but it's not tested and may not work
- Build C++ code using cmake and *Ninja* for **all ABIs**
- Compiled libraries will be in *tools/jniLibs* folder

Command line examples:

```
# to prepare project(choose ABIs which you need)
# for arm64-v8a
cmake -G Ninja -DCMAKE_TOOLCHAIN_FILE=E:\android-ndk-r21d-windows-x86_64\android-ndk-
↪r21d\build\cmake\android.toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-19 -
↪DANDROID_ABI=arm64-v8a ..
# for armeabi-v7a
cmake -G Ninja -DCMAKE_TOOLCHAIN_FILE=E:\android-ndk-r21d-windows-x86_64\android-ndk-
↪r21d\build\cmake\android.toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-19 -
↪DANDROID_ABI=armeabi-v7a ..
# for x86_64
cmake -G Ninja -DCMAKE_TOOLCHAIN_FILE=E:\android-ndk-r21d-windows-x86_64\android-ndk-
↪r21d\build\cmake\android.toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-19 -
↪DANDROID_ABI=x86_64 ..
# for x86
cmake -G Ninja -DCMAKE_TOOLCHAIN_FILE=E:\android-ndk-r21d-windows-x86_64\android-ndk-
↪r21d\build\cmake\android.toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-19 -
↪DANDROID_ABI=x86 ..

# to build(should be run for each ABI from previous step)
cmake --build . --target install --config Release -j 2 --parallel 2
```

CHAPTER

THREE

USER API

BrainFlow User API has three main modules:

- BoardShim to read data from a board, it calls methods from underlying BoardController library
- DataFilter to perform signal processing, it calls methods from underlying DataHandler library
- MLModel to calculate derivative metrics, it calls methods from underlying MLModule library

These classes are independent, so if you want, you can use BrainFlow API only for data streaming and perform signal processing by yourself and vice versa.

BrainFlow data acqusition API is board agnostic, so **to select a specific board you need to pass BrainFlow's board id to BoardShim's constructor and an instance of BrainFlowInputParams structure** which should hold information for your specific board, check *Supported Boards*. for details. This abstraction allows you to switch boards without any changes in code.

In BoardShim, all board data is returned as a 2d array. Rows in this array may contain timestamps, EEG and EMG data and so on. To see instructions how to query specific kind of data check *Data Format Description* and *Code Samples*.

3.1 Python API Reference

3.1.1 brainflow.board_shim

class brainflow.board_shim.**BoardIds**(*value*)

Bases: enum.IntEnum

Enum to store all supported Board Ids

PLAYBACK_FILE_BOARD = -3

STREAMING_BOARD = -2

SYNTHETIC_BOARD = -1

CYTON_BOARD = 0

GANGLION_BOARD = 1

CYTON_DAISY_BOARD = 2

GALEA_BOARD = 3

GANGLION_WIFI_BOARD = 4

CYTON_WIFI_BOARD = 5

CYTON_DAISY_WIFI_BOARD = 6

BRAINBIT_BOARD = 7

UNICORN_BOARD = 8

CALLIBRI_EEG_BOARD = 9

CALLIBRI_EMG_BOARD = 10

CALLIBRI_ECG_BOARD = 11

FASCIA_BOARD = 12

NOTION_OSC_BOARD = 13

NOTION_1_BOARD = 13

NOTION_2_BOARD = 14

IRONBCI_BOARD = 15

GFORCE_PRO_BOARD = 16

FREEEEG32_BOARD = 17

BRAINBIT_BLED_BOARD = 18

GFORCE_DUAL_BOARD = 19

GALEA_SERIAL_BOARD = 20

class brainflow.board_shim.**LogLevels**(*value*)
Bases: enum.IntEnum

Enum to store all log levels supported by BrainFlow

LEVEL_TRACE = 0

LEVEL_DEBUG = 1

LEVEL_INFO = 2

LEVEL_WARN = 3

LEVEL_ERROR = 4

LEVEL_CRITICAL = 5

LEVEL_OFF = 6

class brainflow.board_shim.**IpProtocolType**(*value*)
Bases: enum.IntEnum

Enum to store Ip Protocol types

NONE = 0

UDP = 1

TCP = 2

class brainflow.board_shim.**BrainFlowInputParams**
Bases: object

inputs parameters for prepare_session method

Parameters

- **serial_port** (*str*) – serial port name is used for boards which reads data from serial port

- **mac_address** (*str*) – mac address for example its used for bluetooth based boards
- **ip_address** (*str*) – ip address is used for boards which reads data from socket connection
- **ip_port** (*int*) – ip port for socket connection, for some boards where we know it in front you dont need this parameter
- **ip_protocol** (*int*) – ip protocol type from IpProtocolType enum
- **other_info** (*str*) – other info
- **serial_number** (*str*) – serial number
- **file** (*str*) – file

exception brainflow.board_shim.**BrainFlowError**(*message: str*, *exit_code: int*)

Bases: Exception

This exception is raised if non-zero exit code is returned from C code

Parameters

- **message** (*str*) – exception message
- **exit_code** (*int*) – exit code flow low level API

class brainflow.board_shim.**BoardShim**(*board_id: int*, *input_params: brainflow.board_shim.BrainFlowInputParams*)

Bases: object

BoardShim class is a primary interface to all boards

Parameters

- **board_id** (*int*) – Id of your board
- **input_params** (*BrainFlowInputParams*) – board specific structure to pass required arguments

classmethod set_log_level(*log_level: int*) → None

set BrainFlow log level, use it only if you want to write your own messages to BrainFlow logger, otherwise use enable_board_logger, enable_dev_board_logger or disable_board_logger

Parameters **log_level** (*int*) – log level, to specify it you should use values from LogLevels enum

classmethod enable_board_logger() → None

enable BrainFlow Logger with level INFO, uses stderr for log messages by default

classmethod disable_board_logger() → None

disable BrainFlow Logger

classmethod enable_dev_board_logger() → None

enable BrainFlow Logger with level TRACE, uses stderr for log messages by default

classmethod log_message(*log_level: int*, *message: str*) → None

write your own log message to BrainFlow logger, use it if you wanna have single logger for your own code and BrainFlow's code

Parameters

- **log_level** – log level
- **message** (*str*) – message

classmethod set_log_file(*log_file: str*) → None
redirect logger from stderr to file, can be called any time

Parameters **log_file** (*str*) – log file name

classmethod get_sampling_rate(*board_id: int*) → int
get sampling rate for a board

Parameters **board_id** (*int*) – Board Id

Returns sampling rate for this board id

Return type int

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_package_num_channel(*board_id: int*) → int
get package num channel for a board

Parameters **board_id** (*int*) – Board Id

Returns number of package num channel

Return type int

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_battery_channel(*board_id: int*) → int
get battery channel for a board

Parameters **board_id** (*int*) – Board Id

Returns number of batter channel

Return type int

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_num_rows(*board_id: int*) → int
get number of rows in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns number of rows in returned numpy array

Return type int

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_timestamp_channel(*board_id: int*) → int
get timestamp channel in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns number of timestamp channel in returned numpy array

Return type int

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_marker_channel(*board_id: int*) → int
get marker channel in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns number of marker channel in returned numpy array

Return type int

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_eeg_names(*board_id: int*) → List[str]
get names of EEG channels in 10-20 system if their location is fixed

Parameters **board_id** (*int*) – Board Id

Returns EEG channels names

Return type List[str]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_board_descr(*board_id: int*)
get board description as json

Parameters **board_id** (*int*) – Board Id

Returns info about board

Return type json

Raises **BrainFlowError** – If there is no such board id exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_device_name(*board_id: int*) → str
get device name

Parameters **board_id** (*int*) – Board Id

Returns Device Name

Return type str

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_eeg_channels(*board_id: int*) → List[int]
get list of eeg channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of eeg channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_exg_channels(*board_id: int*) → List[int]
get list of exg channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of eeg channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_emg_channels(*board_id: int*) → List[int]
get list of emg channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of eeg channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_ecg_channels(*board_id: int*) → List[int]
get list of ecg channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of ecg channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_eog_channels(*board_id: int*) → List[int]
get list of eog channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of eog channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_eda_channels(*board_id: int*) → List[int]
get list of eda channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of eda channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_ppg_channels(*board_id: int*) → List[int]
get list of ppg channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of ppg channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_accel_channels(*board_id: int*) → List[int]
get list of accel channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of accel channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_analog_channels(*board_id: int*) → List[int]
get list of analog channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of analog channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_gyro_channels(*board_id: int*) → List[int]
get list of gyro channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of gyro channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_other_channels(*board_id: int*) → List[int]
get list of other channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of other channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_temperature_channels(*board_id: int*) → List[int]
get list of temperature channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of temperature channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

classmethod get_resistance_channels(*board_id: int*) → List[int]
get list of resistance channels in resulting data table for a board

Parameters **board_id** (*int*) – Board Id

Returns list of resistance channels in returned numpy array

Return type List[int]

Raises **BrainFlowError** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

prepare_session() → None
prepare streaming sesssion, init resources, you need to call it before any other BoardShim object methods

start_stream(*num_samples: int = 450000*, *streamer_params: str = None*) → None
Start streaming data, this methods stores data in ringbuffer

Parameters

- **num_samples** (*int*) – size of ring buffer to keep data
- **parameter to stream data from brainflow, supported vals** (*streamer_params*) – "file://%file_name%:w", "file://%file_name%:a", "streaming_board://%multicast_group_ip%:%port%". Range for multicast addresses is from "224.0.0.0" to "239.255.255.255"

stop_stream() → None
Stop streaming data

release_session() → None
release all resources

get_current_board_data()
Get specified amount of data or less if there is not enough data, doesnt remove data from ringbuffer

Parameters **num_samples** (*int*) – max number of samples

Returns latest data from a board

Return type NDArray[Float64]

get_board_data_count() → int
Get num of elements in ringbuffer

Returns number of elements in ring buffer

Return type int

get_board_id() → int
Get's the actual board id, can be different than provided

Returns board id

Return type int

insert_marker(*value: float*) → None
Insert Marker to Data Stream

Parameters **value** (*float*) – value to insert

Returns board id

Return type int

is_prepared() → bool
Check if session is ready or not

Returns session status

Return type bool

get_board_data()
Get all board data and remove them from ringbuffer

Returns all data from a board

Return type NDArray[Float64]

config_board(*config*) → None
Use this method carefully and only if you understand what you are doing, do NOT use it to start or stop streaming

Parameters **config** (*str*) – string to send to a board

Returns response string if any

Return type str

3.1.2 brainflow.exit_codes

class brainflow.exit_codes.**BrainflowExitCodes**(*value*)

Bases: enum.IntEnum

Enum to store all possible exit codes

STATUS_OK = 0

PORT_ALREADY_OPEN_ERROR = 1

UNABLE_TO_OPEN_PORT_ERROR = 2

SER_PORT_ERROR = 3

BOARD_WRITE_ERROR = 4

INCOMMING_MSG_ERROR = 5

INITIAL_MSG_ERROR = 6

BOARD_NOT_READY_ERROR = 7

STREAM_ALREADY_RUN_ERROR = 8

INVALID_BUFFER_SIZE_ERROR = 9

STREAM_THREAD_ERROR = 10

STREAM_THREAD_IS_NOT_RUNNING = 11

EMPTY_BUFFER_ERROR = 12

INVALID_ARGUMENTS_ERROR = 13

UNSUPPORTED_BOARD_ERROR = 14

BOARD_NOT_CREATED_ERROR = 15

ANOTHER_BOARD_IS_CREATED_ERROR = 16

GENERAL_ERROR = 17

SYNC_TIMEOUT_ERROR = 18

JSON_NOT_FOUND_ERROR = 19

NO_SUCH_DATA_IN_JSON_ERROR = 20

CLASSIFIER_IS_NOT_PREPARED_ERROR = 21

ANOTHER_CLASSIFIER_IS_PREPARED_ERROR = 22

UNSUPPORTED_CLASSIFIER_AND_METRIC_COMBINATION_ERROR = 23

3.1.3 brainflow.data_filter

class brainflow.data_filter.**FilterTypes**(*value*)
Bases: enum.IntEnum

Enum to store all supported Filter Types

BUTTERWORTH = 0

CHEBYSHEV_TYPE_1 = 1

BESSEL = 2

class brainflow.data_filter.**AggOperations**(*value*)
Bases: enum.IntEnum

Enum to store all supported aggregation operations

MEAN = 0

MEDIAN = 1

EACH = 2

class brainflow.data_filter.**WindowFunctions**(*value*)
Bases: enum.IntEnum

Enum to store all supported window functions

NO_WINDOW = 0

HANNING = 1

HAMMING = 2

BLACKMAN_HARRIS = 3

class brainflow.data_filter.**DetrendOperations**(*value*)
Bases: enum.IntEnum

Enum to store all supported detrend options

NONE = 0

CONSTANT = 1

LINEAR = 2

class brainflow.data_filter.**NoiseTypes**(*value*)
Bases: enum.IntEnum

Enum to store noise types

class brainflow.data_filter.**DataFilter**
Bases: object

DataFilter class contains methods for signal processig

classmethod enable_data_logger() → None
enable Data Logger with level INFO, uses stderr for log messages by default

classmethod disable_data_logger() → None
disable Data Logger

classmethod enable_dev_data_logger() → None
enable Data Logger with level TRACE, uses stderr for log messages by default

classmethod set_log_file(*log_file: str*) → None
redirect logger from stderr to file, can be called any time

Parameters **log_file** (*str*) – log file name

classmethod perform_lowpass()
apply low pass filter to provided data

Parameters

- **data** (*NDArray[Float64]*) – data to filter, filter works in-place
- **sampling_rate** (*int*) – board's sampling rate
- **cutoff** (*float*) – cutoff frequency
- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum
- **ripple** (*float*) – ripple value for Chebyshev filter

classmethod perform_highpass()
apply high pass filter to provided data

Parameters

- **data** (*NDArray[Float64]*) – data to filter, filter works in-place
- **sampling_rate** (*int*) – board's sampling rate
- **cutoff** (*float*) – cutoff frequency
- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum
- **ripple** (*float*) – ripple value for Chebyshev filter

classmethod perform_bandpass()
apply band pass filter to provided data

Parameters

- **data** (*NDArray[Float64]*) – data to filter, filter works in-place
- **sampling_rate** (*int*) – board's sampling rate
- **center_freq** (*float*) – center frequency
- **band_width** (*float*) – band width
- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum
- **ripple** (*float*) – ripple value for Chebyshev filter

classmethod perform_bandstop()
apply band stop filter to provided data

Parameters

- **data** (*NDArray[Float64]*) – data to filter, filter works in-place
- **sampling_rate** (*int*) – board's sampling rate
- **center_freq** (*float*) – center frequency
- **band_width** (*float*) – band width

- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum
- **ripple** (*float*) – ripple value for Chebyshev filter

classmethod remove_environmental_noise()
remove env noise using notch filter

Parameters

- **data** (*NDArray[Float64]*) – data to filter, filter works in-place
- **sampling_rate** (*int*) – board's sampling rate
- **noise_type** (*int*) – noise type

classmethod perform_rolling_filter()
smooth data using moving average or median

Parameters

- **data** (*NDArray[Float64]*) – data to smooth, it works in-place
- **period** (*int*) – window size
- **operation** (*int*) – int value from AggOperation enum

classmethod perform_downsampling()
perform data downsampling, it doesnt apply lowpass filter for you, it just aggregates several data points

Parameters

- **data** (*NDArray[Float64]*) – initial data
- **period** (*int*) – downsampling period
- **operation** (*int*) – int value from AggOperation enum

Returns downsampled data

Return type NDArray[Float64]

classmethod perform_wavelet_transform()
perform wavelet transform

Parameters

- **data** (*NDArray[Float64]*) – initial data
- **wavelet** (*str*) – supported vals: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2
  ,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- **decomposition_level** (*int*) – level of decomposition

Returns tuple of wavelet coeffs in format [A(J) D(J) D(J-1) . . . .. D(1)] where J is decomposition level, A - app coeffs, D - detailed coeffs, and array with lengths for each block

Return type tuple

classmethod perform_inverse_wavelet_transform()
perform wavelet transform

Parameters

- **wavelet_output** – tuple of wavelet_coeffs and array with lengths
- **original_data_len** (*int*) – len of signal before wavelet transform

- **wavelet** (*str*) – supported vals: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2
,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- **decomposition_level** (*int*) – level of decomposition

Returns restored data

Return type NDArray[Float64]

classmethod perform_wavelet_denoising()
perform wavelet denoising

Parameters

- **data** (*NDArray[Float64]*) – data to denoise
- **wavelet** (*str*) – supported vals: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2
,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- **decomposition_level** (*int*) – decomposition level

classmethod get_csp()
calculate filters and the corresponding eigenvalues using the Common Spatial Patterns

Parameters

- **data** (*NDArray[Float64]*) – [epochs x channels x times]-shaped 3D array of data for two classes
- **labels** (*NDArray[Int64]*) – n_epochs-length 1D array of zeros and ones that assigns class labels for each epoch. Zero corresponds to the first class

Returns [channels x channels]-shaped 2D array of filters and [channels]-length 1D array of the corresponding eigenvalues

Return type Tuple

classmethod get_window()
perform data windowing

Parameters

- **window_function** – window function
- **window_len** – len of the window function

Returns numpy array, len of the array is the same as data

Return type NDArray[Float64]

classmethod perform_fft()
perform direct fft

Parameters

- **data** (*NDArray[Float64]*) – data for fft, len of data must be a power of 2
- **window** (*int*) – window function

Returns numpy array of complex values, len of this array is N / 2 + 1

Return type NDArray[Complex128]

classmethod get_psd()
calculate PSD

Parameters

- **data** (*NDArray[Float64]*) – data to calc psd, len of data must be a power of 2
- **sampling_rate** (*int*) – sampling rate
- **window** (*int*) – window function

Returns amplitude and frequency arrays of len N / 2 + 1

Return type tuple

classmethod get_psd_welch()
calculate PSD using Welch method

Parameters

- **data** (*NDArray[Float64]*) – data to calc psd
- **nfft** (*int*) – FFT Window size, must be power of 2
- **overlap** (*int*) – overlap of FFT Windows, must be between 0 and nfft
- **sampling_rate** (*int*) – sampling rate
- **window** (*int*) – window function

Returns amplitude and frequency arrays of len N / 2 + 1

Return type tuple

classmethod detrend()
detrend data

Parameters

- **data** (*NDArray[Float64]*) – data to calc psd
- **detrend_operation** (*int*) – Type of detrend operation

classmethod get_band_power(*psd: Tuple*, *freq_start: float*, *freq_end: float*) → float
calculate band power

Parameters

- **psd** (*typle*) – psd from get_psd
- **freq_start** (*int*) – start freq
- **freq_end** (*int*) – end freq

Returns band power

Return type float

classmethod get_avg_band_powers(*data: nptyping.types._ndarray.NDArray*, *channels: List*, *sampling_rate: int*, *apply_filter: bool*) → Tuple
calculate avg and stddev of BandPowers across all channels

Parameters

- **data** (*NDArray*) – 2d array for calculation
- **channels** (*List*) – channels - rows of data array which should be used for calculation
- **sampling_rate** (*int*) – sampling rate
- **apply_filter** (*bool*) – apply bandpass and bandstop filtrers or not

Returns avg and stddev arrays for bandpowers

Return type tuple

classmethod perform_ifft()
perform inverse fft

Parameters **data** (*NDArray[Complex128]*) – data from fft

Returns restored data

Return type NDArray[Float64]

classmethod get_nearest_power_of_two(*value: int*) → int
calc nearest power of two

Parameters **value** (*int*) – input value

Returns nearest power of two

Return type int

classmethod write_file(*data*, *file_name: str*, *file_mode: str*) → None
write data to file, in file data will be transposed

Parameters

- **data** (*2d numpy array*) – data to store in a file
- **file_name** (*str*) – file name to store data
- **file_mode** (*str*) – 'w' to rewrite file or 'a' to append data to file

classmethod read_file(*file_name: str*)
read data from file

Parameters **file_name** (*str*) – file name to read

Returns 2d numpy array with data from this file, data will be transposed to original dimensions

Return type 2d numpy array

3.1.4 brainflow.ml_model

class brainflow.ml_model.**BrainFlowMetrics**(*value*)
Bases: enum.IntEnum

Enum to store all supported metrics

RELAXATION = 0

CONCENTRATION = 1

class brainflow.ml_model.**BrainFlowClassifiers**(*value*)
Bases: enum.IntEnum

Enum to store all supported classifiers

REGRESSION = 0

KNN = 1

SVM = 2

LDA = 3

class brainflow.ml_model.**BrainFlowModelParams**(*metric*, *classifier*)
Bases: object

inputs parameters for prepare_session method

Parameters

- **metric** (*int*) – metric to calculate
- **classifier** (*int*) – classifier to use
- **file** (*str*) – file to load model
- **other_info** (*int*) – additional information

class brainflow.ml_model.**MLModel**(*model_params: brainflow.ml_model.BrainFlowModelParams*)
Bases: object

MLModel class used to calc derivative metrics from raw data

Parameters **model_params** (*BrainFlowModelParams*) – Model Params

classmethod enable_ml_logger() → None
enable ML Logger with level INFO, uses stderr for log messages by default

classmethod disable_ml_logger() → None
disable BrainFlow Logger

classmethod enable_dev_ml_logger() → None
enable ML Logger with level TRACE, uses stderr for log messages by default

classmethod set_log_file(*log_file: str*) → None
redirect logger from stderr to file, can be called any time

Parameters **log_file** (*str*) – log file name

prepare() → None
prepare classifier

release() → None
release classifier

predict(*data: nptyping.types._ndarray.NDArray*) → float
calculate metric from data

Parameters **data** (*NDArray*) – input array

Returns metric value

Return type float

3.2 C++ API Reference

3.2.1 BoardShim class

class BoardShim
BoardShim class to communicate with a board.

Public Functions

BoardShim(int *board_id*, **struct** BrainFlowInputParams *params*)

~BoardShim()

void **prepare_session**()
prepare BrainFlow's streaming session, should be called first

void **start_stream**(int *buffer_size* = 450000, std::string *streamer_params* = "")
start streaming thread and store data in ringbuffer

Parameters

- `buffer_size`: size of internal ring buffer
- `streamer_params`: use it to pass data packages further or store them directly during streaming, supported values: "file://%file_name%:w", "file://%file_name%:a", "streaming_board://%multicast_group_ip%:%port%"".

  Range for multicast addresses is from "224.0.0.0" to "239.255.255.255"

bool **is_prepared**()
check if session is ready or not

void **stop_stream**()
stop streaming thread, doesnt release other resources

void **release_session**()
release streaming session

BrainFlowArray<double, 2> **get_current_board_data**(int *num_samples*)
get latest collected data, doesnt remove it from ringbuffer

int **get_board_id**()
Get board id, for some boards can be different than provided (playback, streaming)

int **get_board_data_count**()
get number of packages in ringbuffer

BrainFlowArray<double, 2> **get_board_data**()
get all collected data and flush it from internal buffer

std::string **config_board**(char **config*)
send string to a board, use it carefully and only if you understand what you are doing

void **insert_marker**(double *value*)
insert marker in data stream

Public Members

int **board_id**

Public Static Functions

void **disable_board_logger**()
disable BrainFlow loggers

void **enable_board_logger**()
enable BrainFlow logger with LEVEL_INFO

void **enable_dev_board_logger**()
enable BrainFlow logger with LEVEL_TRACE

void **set_log_file**(std::string *log_file*)
redirect BrainFlow logger from stderr to file

void **set_log_level**(int *log_level*)
use set_log_level only if you want to write your own log messages to BrainFlow logger

void **log_message**(int *log_level*, **const** char **format*, ...)
write user defined string to BrainFlow logger

json **get_board_descr**(int *board_id*)
get board description as json

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If board id is not valid exit code is UNSUPPORTED_BOARD_ERROR

int **get_sampling_rate**(int *board_id*)
get sampling rate for this board

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **get_package_num_channel**(int *board_id*)
get row index which holds package nums

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **get_timestamp_channel**(int *board_id*)
get row index which holds timestamps

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **`get_marker_channel`** (int *board_id*)
get row index which holds markers

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **`get_battery_channel`** (int *board_id*)
get row index which holds battery level info

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **`get_num_rows`** (int *board_id*)
get number of rows in returned from *get_board_data()* 2d array

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::string **`get_device_name`** (int *board_id*)
get device name

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<std::string> **`get_eeg_names`** (int *board_id*)
get eeg channel names in 10-20 system for devices with fixed electrode locations

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **`get_eeg_channels`** (int *board_id*)
get row indices which hold EEG data, for some board we can not split EEG...

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **`get_emg_channels`** (int *board_id*)
get row indices which hold EMG data, for some board we can not split EEG...

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **`get_ecg_channels`** (int *board_id*)
get row indices which hold ECG data, for some board we can not split EEG...

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **`get_eog_channels`** (int *board_id*)
get row indices which hold EOG data, for some board we can not split EEG...

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **`get_exg_channels`** (int *board_id*)
get row indices which hold EXG data

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **`get_ppg_channels`** (int *board_id*)
get row indices which hold PPG data

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **get_eda_channels** (int *board_id*)
get row indices which hold EDA data

Parameters

- board_id: board id of your device

Exceptions

- BrainFlowException: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **get_accel_channels** (int *board_id*)
get row indices which hold accel data

Parameters

- board_id: board id of your device

Exceptions

- BrainFlowException: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **get_analog_channels** (int *board_id*)
get row indices which hold analog data

Parameters

- board_id: board id of your device

Exceptions

- BrainFlowException: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **get_gyro_channels** (int *board_id*)
get row indices which hold gyro data

Parameters

- board_id: board id of your device

Exceptions

- BrainFlowException: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **get_other_channels** (int *board_id*)
get row indices which hold other information

Parameters

- board_id: board id of your device

Exceptions

- BrainFlowException: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **get_temperature_channels** (int *board_id*)
get row indices which hold temperature data

Parameters

- board_id: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

std::vector<int> **get_resistance_channels**(int *board_id*)
get row indices which hold resistance data

Parameters

- `board_id`: board id of your device

Exceptions

- `BrainFlowException`: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

3.2.2 DataFilter class

class DataFilter
DataFilter class to perform signal processing.

Public Static Functions

void **enable_data_logger**()
enable Data logger with LEVEL_INFO

void **disable_data_logger**()
disable Data loggers

void **enable_dev_data_logger**()
enable Data logger with LEVEL_TRACE

void **set_log_level**(int *log_level*)
set log level

void **set_log_file**(std::string *log_file*)
set log file

void **perform_lowpass**(double **data*, int *data_len*, int *sampling_rate*, double *cutoff*, int *order*, int *filter_type*, double *ripple*)
perform low pass filter in-place

void **perform_highpass**(double **data*, int *data_len*, int *sampling_rate*, double *cutoff*, int *order*, int *filter_type*, double *ripple*)
perform high pass filter in-place

void **perform_bandpass**(double **data*, int *data_len*, int *sampling_rate*, double *center_freq*, double *band_width*, int *order*, int *filter_type*, double *ripple*)
perform bandpass filter in-place

void **perform_bandstop**(double **data*, int *data_len*, int *sampling_rate*, double *center_freq*, double *band_width*, int *order*, int *filter_type*, double *ripple*)
perform bandstop filter in-place

void **remove_environmental_noise**(double **data*, int *data_len*, int *sampling_rate*, int *noise_type*)
apply notch filter to remove env noise

void **perform_rolling_filter**(double **data*, int *data_len*, int *period*, int *agg_operation*)
perform moving average or moving median filter in-place

double ***perform_downsampling**(double **data*, int *data_len*, int *period*, int *agg_operation*, int **filtered_size*)

perform data downsampling, it just aggregates several data points

std::pair<double*, int*> ***perform_wavelet_transform**(double **data*, int *data_len*, std::string *wavelet*, int *decomposition_level*)

perform wavelet transform

Return std::pair of wavelet coeffs array in format [A(J) D(J) D(J-1) . . . .. D(1)] where J is decomposition level A - app coeffs, D - detailed coeffs, and array of lengths for each block in wavelet coeffs array, length of this array is decomposition_level + 1

Parameters

- `data`: input array, any size
- `data_len`: length of input array
- `wavelet`: supported vals: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2.4,bior2.6 ,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- `decomposition_level`: level of decomposition in wavelet transform

double ***perform_inverse_wavelet_transform**(std::pair<double*, int*> *wavelet_output*, int *original_data_len*, std::string *wavelet*, int *decomposition_level*)

performs inverse wavelet transform

void **perform_wavelet_denoising**(double **data*, int *data_len*, std::string *wavelet*, int *decomposition_level*)

perform wavelet denoising

std::pair<BrainFlowArray<double, 2>, BrainFlowArray<double, 1>> **get_csp**(**const** BrainFlowArray<double, 3> *&data*, **const** BrainFlowArray<double, 1> *&labels*)

calculate filters and the corresponding eigenvalues using the Common Spatial Patterns

Return pair of two arrays. The first [n_channel x n_channel]-shaped 2D array represents filters. The second n-channel length 1D array represents eigenvalues

Parameters

- `data`: [n_epochs x n_channels x n_times]-shaped 3D array of data for two classes
- `labels`: n_epochs-length 1D array of zeros and ones that assigns class labels for each epoch. Zero corresponds to the first class
- `n_epochs`: the total number of epochs
- `n_channels`: the number of EEG channels
- `n_times`: the number of samples (observations) for a single epoch for a single channel

double ***get_window**(int *window_function*, int *window_len*)

perform data windowing

std::complex<double> ***perform_fft**(double **data*, int *data_len*, int *window*)

perform direct fft

Return complex array with size data_len / 2 + 1, it holds only positive im values

Parameters

- `data`: input array

- `data_len`: must be power of 2
- `window`: window function

double ***perform_ifft**(std::complex<double> *data, int data_len)*
perform inverse fft

Return restored data

Parameters

- `data`: complex array from perform_fft
- `data_len`: len of original array, must be power of 2

int **get_nearest_power_of_two**(int *value*)
calculate nearest power of 2

Return nearest power of 2

Parameters

- `value`: input value

std::pair<double*, double*> **get_psd**(double *data, int *data_len*, int *sampling_rate*, int *window*)
calculate PSD

Return pair of amplitude and freq arrays of size data_len / 2 + 1

Parameters

- `data`: input array
- `data_len`: must be power of 2
- `sampling_rate`: sampling rate
- `window`: window function

void **detrend**(double *data, int *data_len*, int *detrend_operation*)
subtract trend from data

Parameters

- `data`: input array
- `data_len`:
- `detrend_operation`: use DetrendOperations enum

std::pair<double*, double*> **get_psd_welch**(double *data, int *data_len*, int *nfft*, int *overlap*, int *sampling_rate*, int *window*)

double **get_band_power**(std::pair<double*, double*> *psd*, int *data_len*, double *freq_start*, double *freq_end*)
calculate band power

Return band power

Parameters

- `psd`: psd calculated using get_psd
- `data_len`: len of ampl and freq arrays: N / 2 + 1 where N is FFT size
- `freq_start`: lowest frequency
- `freq_end`: highest frequency

std::pair<double*, double*> **get_avg_band_powers**(**const** BrainFlowArray<double, 2> &*data*, std::vector<int> *channels*, int *sampling_rate*, bool *apply_filters*)
calculate avg and stddev of BandPowers across all channels

Return pair of double arrays of size 5, first of them - avg band powers, second stddev

Parameters

- `data`: input 2d array
- `cols`: number of cols in 2d array - number of datapoints
- `channels`: vector of rows - eeg channels which should be used
- `sampling_rate`: sampling rate
- `apply_filters`: set to true to apply filters before band power calculations

void **write_file**(**const** BrainFlowArray<double, 2> &*data*, std::string *file_name*, std::string *file_mode*)
write file, in file data will be transposed

BrainFlowArray<double, 2> **read_file**(std::string *file_name*)
read data from file, data will be transposed to original format

3.2.3 MLModel class

class MLModel
Calculates different metrics from raw data.

Public Functions

MLModel(**struct** BrainFlowModelParams *params*)

~MLModel()

void **prepare**()
initialize classifier, should be called first

double **predict**(double **data*, int *data_len*)
calculate metric from data

void **release**()
release classifier

Public Static Functions

void **set_log_file**(std::string *log_file*)
redirect logger to a file

void **enable_ml_logger**()
enable ML logger with LEVEL_INFO

void **disable_ml_logger**()
disable ML loggers

void **enable_dev_ml_logger**()
enable ML logger with LEVEL_TRACE

void **set_log_level** (int *log_level*)
set log level

3.2.4 BrainFlow constants

```
#pragma once

enum class BrainFlowExitCodes : int
{
    STATUS_OK = 0,
    PORT_ALREADY_OPEN_ERROR = 1,
    UNABLE_TO_OPEN_PORT_ERROR = 2,
    SET_PORT_ERROR = 3,
    BOARD_WRITE_ERROR = 4,
    INCOMMING_MSG_ERROR = 5,
    INITIAL_MSG_ERROR = 6,
    BOARD_NOT_READY_ERROR = 7,
    STREAM_ALREADY_RUN_ERROR = 8,
    INVALID_BUFFER_SIZE_ERROR = 9,
    STREAM_THREAD_ERROR = 10,
    STREAM_THREAD_IS_NOT_RUNNING = 11,
    EMPTY_BUFFER_ERROR = 12,
    INVALID_ARGUMENTS_ERROR = 13,
    UNSUPPORTED_BOARD_ERROR = 14,
    BOARD_NOT_CREATED_ERROR = 15,
    ANOTHER_BOARD_IS_CREATED_ERROR = 16,
    GENERAL_ERROR = 17,
    SYNC_TIMEOUT_ERROR = 18,
    JSON_NOT_FOUND_ERROR = 19,
    NO_SUCH_DATA_IN_JSON_ERROR = 20,
    CLASSIFIER_IS_NOT_PREPARED_ERROR = 21,
    ANOTHER_CLASSIFIER_IS_PREPARED_ERROR = 22,
    UNSUPPORTED_CLASSIFIER_AND_METRIC_COMBINATION_ERROR = 23
};

enum class BoardIds : int
{
    PLAYBACK_FILE_BOARD = -3,
    STREAMING_BOARD = -2,
    SYNTHETIC_BOARD = -1,
    CYTON_BOARD = 0,
    GANGLION_BOARD = 1,
    CYTON_DAISY_BOARD = 2,
    GALEA_BOARD = 3,
    GANGLION_WIFI_BOARD = 4,
    CYTON_WIFI_BOARD = 5,
    CYTON_DAISY_WIFI_BOARD = 6,
    BRAINBIT_BOARD = 7,
    UNICORN_BOARD = 8,
    CALLIBRI_EEG_BOARD = 9,
    CALLIBRI_EMG_BOARD = 10,
    CALLIBRI_ECG_BOARD = 11,
    FASCIA_BOARD = 12,
    NOTION_1_BOARD = 13,
    NOTION_2_BOARD = 14,
    IRONBCI_BOARD = 15,
```

(continues on next page)

(continued from previous page)

```
    GFORCE_PRO_BOARD = 16,
    FREEEEG32_BOARD = 17,
    BRAINBIT_BLED_BOARD = 18,
    GFORCE_DUAL_BOARD = 19,
    GALEA_SERIAL_BOARD = 20,
    // use it to iterate
    FIRST = PLAYBACK_FILE_BOARD,
    LAST = GALEA_SERIAL_BOARD
};

enum class FilterTypes : int
{
    BUTTERWORTH = 0,
    CHEBYSHEV_TYPE_1 = 1,
    BESSEL = 2
};

enum class AggOperations : int
{
    MEAN = 0,
    MEDIAN = 1,
    EACH = 2
};

enum class WindowFunctions : int
{
    NO_WINDOW = 0,
    HANNING = 1,
    HAMMING = 2,
    BLACKMAN_HARRIS = 3
};

enum class DetrendOperations : int
{
    NONE = 0,
    CONSTANT = 1,
    LINEAR = 2
};

enum class BrainFlowMetrics : int
{
    RELAXATION = 0,
    CONCENTRATION = 1
};

enum class BrainFlowClassifiers : int
{
    REGRESSION = 0,
    KNN = 1,
    SVM = 2,
    LDA = 3
};

/// LogLevels enum to store all possible log levels
enum class LogLevels : int
{
    LEVEL_TRACE = 0,     /// TRACE
```

(continues on next page)

(continued from previous page)

```
    LEVEL_DEBUG = 1,     /// DEBUG
    LEVEL_INFO = 2,      /// INFO
    LEVEL_WARN = 3,      /// WARN
    LEVEL_ERROR = 4,     /// ERROR
    LEVEL_CRITICAL = 5, /// CRITICAL
    LEVEL_OFF = 6        // OFF
};

enum class NoiseTypes : int
{
    FIFTY = 0,
    SIXTY = 1
};
```

3.3 Java API Reference

Content of Brainflow Package:

enum AggOperations
enum to store all supported aggregation operations

Public Functions

int **get_code**()

brainflow::AggOperations (final int code)

Public Members

brainflow::MEAN =(0)

brainflow::MEDIAN =(1)

brainflow::EACH =(2)

Public Static Functions

String brainflow::string_from_code (final int code)

AggOperations brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::agg_operation

Private Static Attributes

final Map< Integer, AggOperations > brainflow::ao_map = new HashMap<Integer, AggOpera

enum BoardIds
enum to store all supported boards

Public Functions

int **get_code** ( )

brainflow::BoardIds (final int code)

Public Members

brainflow::PLAYBACK_FILE_BOARD =(-3)

brainflow::STREAMING_BOARD =(-2)

brainflow::SYNTHETIC_BOARD =(-1)

brainflow::CYTON_BOARD =(0)

brainflow::GANGLION_BOARD =(1)

brainflow::CYTON_DAISY_BOARD =(2)

brainflow::GALEA_BOARD =(3)

brainflow::GANGLION_WIFI_BOARD =(4)

brainflow::CYTON_WIFI_BOARD =(5)

brainflow::CYTON_DAISY_WIFI_BOARD =(6)

brainflow::BRAINBIT_BOARD =(7)

brainflow::UNICORN_BOARD =(8)

brainflow::CALLIBRI_EEG_BOARD =(9)

brainflow::CALLIBRI_EMG_BOARD =(10)

brainflow::CALLIBRI_ECG_BOARD =(11)

brainflow::FASCIA_BOARD =(12)

brainflow::NOTION_1_BOARD =(13)

brainflow::NOTION_2_BOARD =(14)

brainflow::IRONBCI_BOARD =(15)

brainflow::GFORCE_PRO_BOARD =(16)

brainflow::FREEEEG32_BOARD =(17)

brainflow::BRAINBIT_BLED_BOARD =(18)

brainflow::GFORCE_DUAL_BOARD =(19)

brainflow::GALEA_SERIAL_BOARD =(20)

Public Static Functions

String brainflow::string_from_code (final int code)

BoardIds brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::board_id

Private Static Attributes

final Map< Integer, BoardIds > brainflow::bi_map = new HashMap<Integer, BoardIds> ()

class brainflow::brainflow::**BoardShim**

BoardShim class to communicate with a board

Public Functions

BoardShim(int *board_id*, *BrainFlowInputParams* *params*)

Create BoardShim object

void **prepare_session**()

prepare steaming session, allocate resources

int **get_board_id**()

Get Board Id, can be different than provided (playback or streaming board)

String **config_board**(String *config*)

send string to a board, use this method carefully and only if you understand what you are doing

void **start_stream**(int *buffer_size*, String *streamer_params*)

start streaming thread, store data in internal ringbuffer and stream them from brainflow at the same time

Parameters

- buffer_size: size of internal ringbuffer
- streamer_params: supported vals: "file://%file_name%:w", "file://%file_name%:a", "streaming_board://%multicast_group_ip%:%port%". Range for multicast addresses is from "224.0.0.0" to "239.255.255.255"

void **start_stream**()

start streaming thread, store data in internal ringbuffer

void **start_stream**(int *buffer_size*)

start streaming thread, store data in internal ringbuffer

void **stop_stream**()

stop streaming thread

void **release_session**()

release all resources

int **get_board_data_count** ()
get number of packages in ringbuffer

void **insert_marker** (double *value*)
insert marker to data stream

boolean **is_prepared** ()
check session status

double [][] get_current_board_data (int num_samples)
get latest collected data, can return less than "num_samples", doesnt flush it from ringbuffer

double [][] get_board_data ()
get all data from ringbuffer and flush it

Public Members

int **board_id**
BrainFlow's board id

Public Static Functions

void **enable_board_logger** ()
enable BrainFlow logger with level INFO

void **enable_dev_board_logger** ()
enable BrainFlow logger with level TRACE

void **disable_board_logger** ()
disable BrainFlow logger

void **set_log_file** (String *log_file*)
redirect logger from stderr to a file

void **set_log_level** (int *log_level*)
set log level

void **log_message** (int *log_level*, String *message*)
send user defined strings to BrainFlow logger

int **get_sampling_rate** (int *board_id*)
get sampling rate for this board

int **get_timestamp_channel** (int *board_id*)
get row index in returned by *get_board_data()* 2d array which contains timestamps

int **get_marker_channel** (int *board_id*)
get row index in returned by *get_board_data()* 2d array which contains markers

int **get_num_rows** (int *board_id*)
get number of rows in returned by *get_board_data()* 2d array

int **get_package_num_channel** (int *board_id*)
get row index in returned by *get_board_data()* 2d array which contains package nums

int **get_battery_channel** (int *board_id*)
get row index in returned by *get_board_data()* 2d array which contains battery level

String [] get_eeg_names (int board_id)
Get names of EEG electrodes in 10-20 system. Only if electrodes have freezed locations

Map<String, Object> **get_board_descr** (int *board_id*)
Get board description

String **get_device_name** (int *board_id*)
Get device name

int [] get_eeg_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain EEG data, for some boards we can not split EEG... and return the same array

int [] get_emg_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain EMG data, for some boards we can not split EEG... and return the same array

int [] get_ecg_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain ECG data, for some boards we can not split EEG... and return the same array

int [] get_temperature_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain temperature data

int [] get_resistance_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain resistance data

int [] get_eog_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain EOG data, for some boards we can not split EEG... and return the same array

int [] get_exg_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain EXG data

int [] get_eda_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain EDA data, for some boards we can not split EEG... and return the same array

int [] get_ppg_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain PPG data, for some boards we can not split EEG... and return the same array

int [] get_accel_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain accel data

int [] get_analog_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain analog data

int [] get_gyro_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain gyro data

int [] get_other_channels (int board_id)
get row indices in returned by *get_board_data()* 2d array which contain other data

enum BrainFlowClassifiers

Public Functions

int **get_code**()

brainflow::BrainFlowClassifiers (final int code)

Public Members

brainflow::REGRESSION =(0)

brainflow::KNN =(1)

brainflow::SVM =(2)

brainflow::LDA =(3)

Public Static Functions

String brainflow::string_from_code (final int code)

BrainFlowClassifiers brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::protocol

Private Static Attributes

final Map< Integer, BrainFlowClassifiers > brainflow::cl_map = new HashMap<Integer,

class brainflow::brainflow::**BrainFlowError** : **public** Exception

BrainFlowError exception to notify about errors

Public Functions

BrainFlowError(String *message*, int *ec*)

Public Members

String **msg**

int **exit_code**

exit code returned from low level API

class brainflow::brainflow::**BrainFlowInputParams**

to get fields which are required for your board check SupportedBoards section

Public Functions

BrainFlowInputParams()

String **to_json**()

String **get_ip_address**()

void **set_ip_address**(String *ip_address*)

String **get_mac_address**()

void **set_mac_address**(String *mac_address*)

String **get_serial_port**()

void **set_serial_port**(String *serial_port*)

int **get_ip_port**()

void **set_ip_port**(int *ip_port*)

int **get_ip_protocol**()

void **set_ip_protocol**(int *ip_protocol*)

String **get_other_info**()

void **set_other_info**(String *other_info*)

void **set_timeout**(int *timeout*)

int **get_timeout**()

String **get_serial_number**()

void **set_serial_number**(String *serial_number*)

String **get_file**()

void **set_file**(String *file*)

Public Members

String **ip_address**

String **mac_address**

String **serial_port**

int **ip_port**

int **ip_protocol**

String **other_info**

int **timeout**

String **serial_number**

String **file**

enum BrainFlowMetrics

Public Functions

int **get_code**()

brainflow::BrainFlowMetrics (final int code)

Public Members

brainflow::RELAXATION =(0)

brainflow::CONCENTRATION =(1)

Public Static Functions

String brainflow::string_from_code (final int code)

BrainFlowMetrics brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::protocol

Private Static Attributes

final Map< Integer, BrainFlowMetrics > brainflow::metr_map = new HashMap<Integer, Br

class brainflow::brainflow::**BrainFlowModelParams**

describe model parameters

Public Functions

BrainFlowModelParams(int *metric*, int *classifier*)

int **get_metric**()

void **set_metric**(int *metric*)

int **get_classifier**()

void **set_classifier**(int *classifier*)

String **get_file**()

void **set_file**(String *file*)

String **get_other_info**()

void **set_other_info**(String *other_info*)

String **to_json**()

Public Members

int **metric**

int **classifier**

String **file**

String **other_info**

class brainflow::brainflow::**DataFilter**
DataFilter class to perform signal processing

Public Static Functions

void **enable_data_logger**()
enable Data logger with level INFO

void **enable_dev_data_logger**()
enable Data logger with level TRACE

void **disable_data_logger**()
disable Data logger

void **set_log_file**(String *log_file*)
redirect logger from stderr to a file

void perform_lowpass (double[] data, int sampling_rate, double cutoff, int order, int
perform lowpass filter in-place

void perform_highpass (double[] data, int sampling_rate, double cutoff, int order, int
perform highpass filter in-place

void perform_bandpass (double[] data, int sampling_rate, double center_freq, double ba
perform bandpass filter in-place

void perform_bandstop (double[] data, int sampling_rate, double center_freq, double ba
perform bandstop filter in-place

void perform_rolling_filter (double[] data, int period, int operation)
perform moving average or moving median filter in-place

void detrend (double[] data, int operation)
subtract trend from data in-place

double [] perform_downsampling (double[] data, int period, int operation)
perform data downsampling, it doesnt apply lowpass filter for you, it just aggregates several data points

void remove_environmental_noise (double[] data, int sampling_rate, int noise_type)
removes noise using notch filter

void perform_wavelet_denoising (double[] data, String wavelet, int decomposition_level
perform wavelet based denoising in-place

Parameters

- `wavelet`: supported vals: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2.4,bior2.6 ,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- `decomposition_level`: level of decomposition of wavelet transform

Pair< double[], int[]> perform_wavelet_transform (double[] data, String wavelet, int d
perform wavelet transform

Parameters

- `wavelet`: supported vals: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2.4,bior2.6 ,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8

double [] perform_inverse_wavelet_transform (Pair< double[], int[]> wavelet_output, int
perform inverse wavelet transform

Pair< double[][], double[]> get_csp (double[][][] data, double[] labels)
get common spatial filters

double [] get_window (int window, int window_len)
perform data windowing

Return array of the size specified in window_len

Parameters

- `window`: window function
- `window_len`: lenght of the window function

Complex [] perform_fft (double[] data, int start_pos, int end_pos, int window)
perform direct fft

Return array of complex values with size N / 2 + 1

Parameters

- `data`: data for fft transform
- `start_pos`: starting position to calc fft
- `end_pos`: end position to calc fft, total_len must be a power of two
- `window`: window function

double [] perform_ifft (Complex[] data)
perform inverse fft

Return restored data

Parameters

- `data`: data from fft transform(array of complex values)

Pair< double[], double[]> get_avg_band_powers (double[][] data, int[] channels, int sa
calc average and stddev of band powers across all channels

Return pair of avgs and stddevs for bandpowers

Parameters

- `data`: data to process
- `channels`: rows of data arrays which should be used in calculation
- `sampling_rate`: sampling rate
- `apply_filters`: apply bandpass and bandstop filters before calculation

Pair< double[], double[]> get_psd (double[] data, int start_pos, int end_pos, int samp
get PSD

Return pair of ampl and freq arrays with len N / 2 + 1

Parameters

- `data`: data to process
- `start_pos`: starting position to calc PSD
- `end_pos`: end position to calc PSD, total_len must be a power of two
- `sampling_rate`: sampling rate
- `window`: window function

Pair< double[], double[]> get_psd_welch (double[] data, int nfft, int overlap, int sam
get PSD using Welch Method

Return pair of ampl and freq arrays

Parameters

- `data`: data to process
- `nfft`: size of FFT, must be power of two
- `overlap`: overlap between FFT Windows, must be between 0 and nfft
- `sampling_rate`: sampling rate
- `window`: window function

double **get_band_power** (Pair<double[], double[]> *psd*, double *freq_start*, double *freq_end*)
get band power

Return band power

Parameters

- `psd`: PSD from get_psd or get_log_psd
- `freq_start`: lowest frequency of band
- `freq_end`: highest frequency of band

int **get_nearest_power_of_two** (int *value*)
calculate nearest power of two

void write_file (double[][] data, String file_name, String file_mode)
write data to tsv file, in file data will be transposed

double [][] read_file (String file_name)
read data from file, transpose it back to original format

enum DetrendOperations
enum to store all supported detrend operations

Public Functions

int **get_code** ( )

brainflow::DetrendOperations (final int code)

Public Members

brainflow::NONE =(0)

brainflow::CONSTANT =(1)

brainflow::LINEAR =(2)

Public Static Functions

String brainflow::string_from_code (final int code)

DetrendOperations brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::detrend_operation

Private Static Attributes

final Map< Integer, DetrendOperations > brainflow::dt_map = new HashMap<Integer, Det

enum ExitCode

Public Functions

int **get_code** ( )

brainflow::ExitCode (final int code)

Public Members

brainflow::STATUS_OK =(0)

brainflow::PORT_ALREADY_OPEN_ERROR =(1)

brainflow::UNABLE_TO_OPEN_PORT_ERROR =(2)

brainflow::SET_PORT_ERROR =(3)

brainflow::BOARD_WRITE_ERROR =(4)

brainflow::INCOMMING_MSG_ERROR =(5)

brainflow::INITIAL_MSG_ERROR =(6)

brainflow::BOARD_NOT_READY_ERROR =(7)

brainflow::STREAM_ALREADY_RUN_ERROR =(8)

brainflow::INVALID_BUFFER_SIZE_ERROR =(9)

brainflow::STREAM_THREAD_ERROR =(10)

brainflow::STREAM_THREAD_IS_NOT_RUNNING =(11)

brainflow::EMPTY_BUFFER_ERROR =(12)

brainflow::INVALID_ARGUMENTS_ERROR =(13)

brainflow::UNSUPPORTED_BOARD_ERROR =(14)

brainflow::BOARD_NOT_CREATED_ERROR =(15)

brainflow::ANOTHER_BOARD_IS_CREATED_ERROR =(16)

brainflow::GENERAL_ERROR =(17)

brainflow::SYNC_TIMEOUT_ERROR =(18)

brainflow::JSON_NOT_FOUND_ERROR =(19)

brainflow::NO_SUCH_DATA_IN_JSON_ERROR =(20)

brainflow::CLASSIFIER_IS_NOT_PREPARED_ERROR =(21)

brainflow::ANOTHER_CLASSIFIER_IS_PREPARED_ERROR =(22)

brainflow::UNSUPPORTED_CLASSIFIER_AND_METRIC_COMBINATION_ERROR =(23)

Public Static Functions

String brainflow::string_from_code (final int code)

ExitCode brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::exit_code

Private Static Attributes

final Map< Integer, ExitCode > brainflow::ec_map = new HashMap<Integer, ExitCode> ()

enum FilterTypes

enum to store all possible filter types

Public Functions

int **get_code** ( )

brainflow::FilterTypes (final int code)

Public Members

brainflow::BUTTERWORTH =(0)

brainflow::CHEBYSHEV_TYPE_1 =(1)

brainflow::BESSEL =(2)

Public Static Functions

String brainflow::string_from_code (final int code)

FilterTypes brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::filter_type

Private Static Attributes

final Map< Integer, FilterTypes > brainflow::ft_map = new HashMap<Integer, FilterTyp

enum IpProtocolType

Public Functions

int **get_code** ( )

brainflow::IpProtocolType (final int code)

Public Members

brainflow::NONE =(0)

brainflow::UDP =(1)

brainflow::TCP =(2)

Public Static Functions

String brainflow::string_from_code (final int code)

IpProtocolType brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::protocol

Private Static Attributes

final Map< Integer, IpProtocolType > brainflow::ip_map = new HashMap<Integer, IpProt

enum LogLevels

Public Functions

int **get_code**()

brainflow::LogLevels (final int code)

Public Members

brainflow::LEVEL_TRACE =(0)

brainflow::LEVEL_DEBUG =(1)

brainflow::LEVEL_INFO =(2)

brainflow::LEVEL_WARN =(3)

brainflow::LEVEL_ERROR =(4)

brainflow::LEVEL_CRITICAL =(5)

brainflow::LEVEL_OFF =(6)

Public Static Functions

String brainflow::string_from_code (final int code)

LogLevels brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::log_level

Private Static Attributes

final Map< Integer, LogLevels > brainflow::ll_map = new HashMap<Integer, LogLevels>

class brainflow::brainflow::**MLModel**

Public Functions

MLModel (*BrainFlowModelParams params*)
Create MLModel object

void **prepare** ()
Prepare classifier

Exceptions

- *BrainFlowError*:

void **release** ()
Release classifier

Exceptions

- *BrainFlowError*:

double predict (double[] data)
Get score of classifier

Exceptions

- *BrainFlowError*:

Public Static Functions

void **enable_ml_logger** ()
enable ML logger with level INFO

void **enable_dev_ml_logger** ()
enable ML logger with level TRACE

void **disable_ml_logger** ()
disable BrainFlow logger

void **set_log_file** (String *log_file*)
redirect logger from stderr to a file

enum NoiseTypes
enum to store all supported noise types

Public Functions

int **get_code** ()

brainflow::NoiseTypes (final int code)

Public Members

brainflow::FIFTY =(0)

brainflow::SIXTY =(1)

brainflow::EACH =(2)

Public Static Functions

String brainflow::string_from_code (final int code)

NoiseTypes brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

final int brainflow::noise_type

Private Static Attributes

final Map< Integer, NoiseTypes > brainflow::nt_map = new HashMap<Integer, NoiseTypes

enum WindowFunctions

Public Functions

int **get_code** ( )

brainflow::WindowFunctions (final int code)

Public Members

brainflow::NO_WINDOW =(0)

brainflow::HANNING =(1)

brainflow::HAMMING =(2)

brainflow::BLACKMAN_HARRIS =(3)

Public Static Functions

String brainflow::string_from_code (final int code)

WindowFunctions brainflow::from_code (final int code)

brainflow::[static initializer]

Private Members

```
final int brainflow::window
```

Private Static Attributes

```
final Map< Integer, WindowFunctions > brainflow::window_map   = new HashMap<Integer, W
```

3.4 C# API Reference

Content of brainflow namespace:

enum brainflow::**LogLevels**

Values:

enumerator LEVEL_TRACE = 0

enumerator LEVEL_DEBUG = 1

enumerator LEVEL_INFO = 2

enumerator LEVEL_WARN = 3

enumerator LEVEL_ERROR = 4

enumerator LEVEL_CRITICAL = 5

enumerator LEVEL_OFF = 6

enum brainflow::**CustomExitCodes**

Values:

enumerator STATUS_OK = 0

enumerator PORT_ALREADY_OPEN_ERROR = 1

enumerator UNABLE_TO_OPEN_PORT_ERROR = 2

enumerator SET_PORT_ERROR = 3

enumerator BOARD_WRITE_ERROR = 4

enumerator INCOMMING_MSG_ERROR = 5

enumerator INITIAL_MSG_ERROR = 6

enumerator BOARD_NOT_READY_ERROR = 7

enumerator STREAM_ALREADY_RUN_ERROR = 8

enumerator INVALID_BUFFER_SIZE_ERROR = 9

enumerator STREAM_THREAD_ERROR = 10

enumerator STREAM_THREAD_IS_NOT_RUNNING = 11

enumerator EMPTY_BUFFER_ERROR = 12

enumerator INVALID_ARGUMENTS_ERROR = 13

enumerator UNSUPPORTED_BOARD_ERROR = 14

enumerator BOARD_NOT_CREATED_ERROR = 15

enumerator ANOTHER_BOARD_IS_CREATED_ERROR = 16

enumerator GENERAL_ERROR = 17

enumerator SYNC_TIMEOUT_ERROR = 18

enumerator JSON_NOT_FOUND_ERROR = 19

enumerator NO_SUCH_DATA_IN_JSON_ERROR = 20

enumerator CLASSIFIER_IS_NOT_PREPARED_ERROR = 21

enumerator ANOTHER_CLASSIFIER_IS_PREPARED_ERROR = 22

enumerator UNSUPPORTED_CLASSIFIER_AND_METRIC_COMBINATION_ERROR = 23

enum brainflow::**BoardIds**

Values:

enumerator PLAYBACK_FILE_BOARD = -3

enumerator STREAMING_BOARD = -2

enumerator SYNTHETIC_BOARD = -1

enumerator CYTON_BOARD = 0

enumerator GANGLION_BOARD = 1

enumerator CYTON_DAISY_BOARD = 2

enumerator GALEA_BOARD = 3

enumerator GANGLION_WIFI_BOARD = 4

enumerator CYTON_WIFI_BOARD = 5

enumerator CYTON_DAISY_WIFI_BOARD = 6

enumerator BRAINBIT_BOARD = 7

enumerator UNICORN_BOARD = 8

enumerator CALLIBRI_EEG_BOARD = 9

enumerator CALLIBRI_EMG_BOARD = 10

enumerator CALLIBRI_ECG_BOARD = 11

enumerator FASCIA_BOARD = 12

enumerator NOTION_1_BOARD = 13

enumerator NOTION_2_BOARD = 14

enumerator IRONBCI_BOARD = 15

enumerator GFORCE_PRO_BOARD = 16

enumerator FREEEEG32_BOARD = 17

enumerator BRAINBIT_BLED_BOARD = 18

enumerator GFORCE_DUAL_BOARD = 19

enumerator GALEA_SERIAL_BOARD = 20

enum brainflow::**IpProtocolType**

Values:

enumerator NONE = 0

enumerator UDP = 1

enumerator TCP = 2

enum brainflow::**FilterTypes**
Values:

enumerator BUTTERWORTH = 0

enumerator CHEBYSHEV_TYPE_1 = 1

enumerator BESSEL = 2

enum brainflow::**AggOperations**
Values:

enumerator MEAN = 0

enumerator MEDIAN = 1

enumerator EACH = 2

enum brainflow::**WindowFunctions**
Values:

enumerator NO_WINDOW = 0

enumerator HANNING = 1

enumerator HAMMING = 2

enumerator BLACKMAN_HARRIS = 3

enum brainflow::**DetrendOperations**
Values:

enumerator NONE = 0

enumerator CONSTANT = 1

enumerator LINEAR = 2

enum brainflow::**NoiseTypes**
Values:

enumerator FIFTY = 0

enumerator SIXTY = 1

enum brainflow::**BrainFlowMetrics**
Values:

enumerator RELAXATION = 0

enumerator CONCENTRATION = 1

enum brainflow::**BrainFlowClassifiers**
Values:

enumerator REGRESSION = 0

enumerator KNN = 1

enumerator SVM = 2

enumerator LDA = 3

class brainflow::brainflow::**BoardDescr**

Public Functions

BoardDescr()

Public Members

int **sampling_rate**

int [] eeg_channels

int [] eog_channels

int [] exg_channels

int [] emg_channels

int [] ppg_channels

int [] eda_channels

int [] accel_channels

int [] gyro_channels

int [] temperature_channels

int [] resistance_channels

int [] other_channels

int **package_num_channel**

int **batter_channel**

int **timestamp_channel**

int **marker_channel**

int **num_rows**

string **name**

string **eeg_names**

class brainflow::brainflow::**BoardShim**

BoardShim class to communicate with a board

Public Functions

BoardShim(int *board_id*, *BrainFlowInputParams* *input_params*)

Create an instance of BoardShim class

Parameters

- board_id
- input_params

void **prepare_session**()

prepare BrainFlow's streaming session, allocate required resources

string **config_board**(string *config*)

send string to a board, use this method carefully and only if you understand what you are doing

Parameters

- `config`

void **insert_marker**(double *value*)
insert marker to data array

void **start_stream**(int *buffer_size* = 3600 * 250, string *streamer_params* = "")
start streaming thread, store data in internal ringbuffer

Parameters

- `buffer_size`: size of internal ringbuffer
- `streamer_params`: supported values: file://%file_name%:w, file://%file_name%:a, streaming_board://multicast_group_ip%:port%

void **stop_stream**()
stop streaming thread, doesnt release other resources

void **release_session**()
release BrainFlow's session

bool **is_prepared**()
check session status

summary> Get Board Id, for some boards can be different than provided /summary>

Return session status

Return Master board id

int **get_board_id**()

int **get_board_data_count**()
get number of packages in ringbuffer

Return number of packages

double [,] get_current_board_data (int num_samples)
get latest collected data, doesnt remove it from ringbuffer

Return latest collected data, can be less than "num_samples"

Parameters

- `num_samples`

double [,] get_board_data ()
get all collected data and remove it from ringbuffer

Return all collected data

Public Members

int **board_id**
BrainFlow's board id

Public Static Functions

int **get_sampling_rate** (int *board_id*)
get sampling rate for this board id

Return sampling rate

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **get_package_num_channel** (int *board_id*)
get row index in returned by *get_board_data()* 2d array which holds package nums

Return row num in 2d array

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **get_timestamp_channel** (int *board_id*)
get row index which holds timestamps

Return row num in 2d array

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **get_marker_channel** (int *board_id*)
get row index which holds marker

Return row num in 2d array

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **get_battery_channel** (int *board_id*)
get row undex which holds battery level

Return row num in 2d array

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int **get_num_rows** (int *board_id*)
get number of rows in returned by *get_board_data()* 2d array

Return number of rows in 2d array

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

string [] get_eeg_names (int board_id)
get names of EEG channels in 10-20 system. Only if electrodes have fixed locations

Return array of 10-20 locations

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

template<>
T **get_board_descr**<T> (int *board_id*)
get board description

Return board description

Parameters

- board_id

Exceptions

- *BrainFlowException*: If board id is not valid exit code is UNSUPPORTED_BOARD_ERROR

string **get_device_name** (int *board_id*)
get device name

Return device name

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_eeg_channels (int board_id)

get row indices of EEG channels for this board, for some board we can not split EMG.. data and return the same array for all of them

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_emg_channels (int board_id)

get row indices of EMG channels for this board, for some board we can not split EMG.. data and return the same array for all of them

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_ecg_channels (int board_id)

get row indices of ECG channels for this board, for some board we can not split EMG.. data and return the same array for all of them

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_eog_channels (int board_id)

get row indices of EOG channels for this board, for some board we can not split EMG.. data and return the same array for all of them

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_exg_channels (int board_id)
get row indices of EXG channels for this board

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_eda_channels (int board_id)
get row indices of EDA channels for this board, for some board we can not split EMG.. data and return the same array for all of them

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_ppg_channels (int board_id)
get row indeces which hold ppg data

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_accel_channels (int board_id)
get row indices which hold accel data

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_analog_channels (int board_id)
get row indices which hold analog data

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_gyro_channels (int board_id)
get row indices which hold gyro data

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_other_channels (int board_id)
get other channels for this board

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_temperature_channels (int board_id)
get temperature channels for this board

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

int [] get_resistance_channels (int board_id)
get resistance channels for this board

Return array of row nums

Parameters

- board_id

Exceptions

- *BrainFlowException*: If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

void **set_log_level**(int *log_level*)
set log level, logger is disabled by default

Parameters

- log_level

void **enable_board_logger**()
enable BrainFlow's logger with level INFO

void **disable_board_logger**()
disable BrainFlow's logger

void **enable_dev_board_logger**()
enable BrainFLow's logger with level TRACE

void **set_log_file**(string *log_file*)
redirect BrainFlow's logger from stderr to file

Parameters

- log_file

void **log_message**(int *log_level*, string *message*)
send your own log message to BrainFlow's logger

Parameters

- log_level
- message

class brainflow::brainflow::**BrainFlowException** : **public** Exception
BrainFlowException class to notify about errors

Public Functions

BrainFlowException(int *code*)

Public Members

int **exit_code**
exit code returned from low level API

class brainflow::brainflow::**BrainFlowInputParams**
Check SuportedBoards to get information about fields which are required for specific board

Public Functions

BrainFlowInputParams()

string **to_json**()

Public Members

string **serial_port**
serial port name

string **mac_address**
MAC address

string **ip_address**
IP address

int **ip_port**
PORT

int **ip_protocol**
IP protocol, use IpProtocolType

string **other_info**
you can provide additional info to low level API using this field

int **timeout**
timeout for device discovery or connection

string **serial_number**
serial number

string **file**
file

class brainflow::brainflow::**BrainFlowModelParams**
Describe model

Public Functions

BrainFlowModelParams(int *metric*, int *classifier*)

string **to_json**()

Public Members

int **metric**
metric to caclulate

int **classifier**
classifier to use

string **file**
path to model file

string **other_info**
other info

class brainflow::brainflow::**DataFilter**
DataFilter class to perform signal processing

Public Static Functions

void **enable_data_logger**()
enable Data logger with level INFO

void **disable_data_logger**()
disable Data logger

void **enable_dev_data_logger**()
enable Data logger with level TRACE

void **set_log_file**(string *log_file*)
redirect BrainFlow's logger from stderr to file

Parameters

- log_file

double [] perform_lowpass (double[] data, int sampling_rate, double cutoff, int order,
perform lowpass filter, unlike other bindings instead in-place calculation it returns new array

Return filtered data

Parameters

- data
- sampling_rate
- cutoff
- order
- filter_type
- ripple

double [] remove_environmental_noise (double[] data, int sampling_rate, int noise_type
remove env noise using notch filter

Return filtered data

Parameters

- data
- sampling_rate
- noise_type

double [] perform_highpass (double[] data, int sampling_rate, double cutoff, int order
perform highpass filter, unlike other bindings instead in-place calculation it returns new array

Return filtered data

Parameters

- data
- sampling_rate

- cutoff
- order
- filter_type
- ripple

double [] perform_bandpass (double[] data, int sampling_rate, double center_freq, doub
perform bandpass filter, unlike other bindings instead in-place calculation it returns new array

Return filtered data

Parameters

- data
- sampling_rate
- center_freq
- band_width
- order
- filter_type
- ripple

double [] perform_bandstop (double[] data, int sampling_rate, double center_freq, doub
perform bandstop filter, unlike other bindings instead in-place calculation it returns new array

Return filtered data

Parameters

- data
- sampling_rate
- center_freq
- band_width
- order
- filter_type
- ripple

double [] perform_rolling_filter (double[] data, int period, int operation)
perform moving average or moving median filter, unlike other bindings instead in-place calculation it returns new array

Return filered data

Parameters

- data
- period
- operation

double [] detrend (double[] data, int operation)
detrend, unlike other bindings instead in-place calculation it returns new array

Return data with removed trend

Parameters

- `data`
- `operation`

double [] perform_downsampling (double[] data, int period, int operation)
perform data downsampling, it just aggregates data without applying lowpass filter

Return data after downsampling

Parameters

- `data`
- `period`
- `operation`

Tuple< double[], int[]> perform_wavelet_transform (double[] data, string wavelet, int
perform wavelet transform

Return tuple of wavelet coeffs in format [A(J) D(J) D(J-1) ….. D(1)] where J is decomposition level, A - app coeffs, D - detailed coeffs, and array with lengths for each block

Parameters

- `data`: data for wavelet transform
- `wavelet`: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2.4,bior2.6,bior2.8,bior3.1 ,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- `decomposition_level`: decomposition level

double [] perform_inverse_wavelet_transform (Tuple< double[], int[]> wavelet_data, int
perform inverse wavelet transorm

Return restored data

Parameters

- `wavelet_data`: tuple returned by perform_wavelet_transform
- `original_data_len`: size of original data before direct wavelet transform
- `wavelet`: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2.4,bior2.6,bior2.8,bior3.1 ,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- `decomposition_level`: level of decomposition

double [] perform_wavelet_denoising (double[] data, string wavelet, int decomposition_
perform wavelet based denoising

Return denoised data

Parameters

- `data`: data for denoising
- `wavelet`: db1..db15,haar,sym2..sym10,coif1..coif5,bior1.1,bior1.3,bior1.5,bior2.2,bior2.4,bior2.6,bior2.8,bior3.1 ,bior3.7,bior3.9,bior4.4,bior5.5,bior6.8
- `decomposition_level`: level of decomposition in wavelet transform

Tuple< double[,], double[]> get_csp (double[,,] data, double[] labels)
get common spatial patterns

Return Tuple of two arrays: [n_channels x n_channels] shaped array of filters and n_channels length array of eigenvalues

Parameters

- `data`: data for csp
- `labels`: labels for each class

double [] get_window (int window_function, int window_len)
perform windowing

Return array of the size specified in window_len

Parameters

- `window_function`: window function
- `window_len`: len of the window

Complex [] perform_fft (double[] data, int start_pos, int end_pos, int window)
perform direct fft

Return complex array of size N / 2 + 1 of fft data

Parameters

- `data`: data for fft
- `start_pos`: start pos
- `end_pos`: end pos, end_pos - start_pos must be a power of 2
- `window`: window function

double [] perform_ifft (Complex[] data)
perform inverse fft

Return restored data

Parameters

- `data`: data from perform_fft

void write_file (double[,] data, string file_name, string file_mode)
write data to tsv file, data will be transposed

Parameters

- `data`
- `file_name`
- `file_mode`

double [,] read_file (string file_name)
read data from file, data will be transposed back to original format

Return

Parameters

- `file_name`

int **`get_nearest_power_of_two`** (int *value*)
calculate nearest power of two

Return nearest power of two

Parameters

- `value`

`Tuple< double[], double[]> get_avg_band_powers (double[,] data, int[] channels, int sa`
calculate avg and stddev bandpowers across channels

Return Tuple of avgs and stddev arrays

Parameters

- `data`: 2d array with values
- `channels`: rows of data array which should be used for calculation
- `sampling_rate`: sampling rate
- `apply_filters`: apply bandpass and bandstop filters before calculation

`Tuple< double[], double[]> get_psd (double[] data, int start_pos, int end_pos, int sam`
calculate PSD

Return Tuple of ampls and freqs arrays of size N / 2 + 1

Parameters

- `data`: data for PSD
- `start_pos`: start pos
- `end_pos`: end pos, end_pos - start_pos must be a power of 2
- `sampling_rate`: sampling rate
- `window`: window function

`Tuple< double[], double[]> get_psd_welch (double[] data, int nfft, int overlap, int sa`
calculate PSD using Welch method

Return Tuple of ampls and freqs arrays

Parameters

- `data`: data for log PSD
- `nfft`: FFT Size
- `overlap`: FFT Window overlap, must be between 0 and nfft
- `sampling_rate`: sampling rate
- `window`: window function

double **`get_band_power`** (Tuple<double[], double[]> *psd*, double *start_freq*, double *stop_freq*)
calculate band power

Return band power

Parameters

- `psd`: psd data returned by get_psd or get_psd_welch
- `start_freq`: lowest frequency of band
- `stop_freq`: highest frequency of band

class brainflow::brainflow::**MLModel**

Public Functions

MLModel(*BrainFlowModelParams input_params*)
Create an instance of MLModel class

Parameters

- `input_params`

void **prepare**()
Prepare classifier

void **release**()
Release classifier

double predict (double[] data)
Get score of classifier

Public Static Functions

void **enable_ml_logger**()
enable ML logger with level INFO

void **disable_ml_logger**()
disable ML logger

void **enable_dev_ml_logger**()
enable ML logger with level TRACE

void **set_log_file**(string *log_file*)
redirect BrainFlow's logger from stderr to file

Parameters

- `log_file`

class brainflow::brainflow::**PlatformHelper**

Public Static Functions

LibraryEnvironment **get_library_environment**()

3.5 R API Reference

R binding is a wrapper on top of Python binding. It is implemented using reticulate.

Check R samples to see how to use it.

Full code for R binding:

```
#' @import reticulate
NULL

#' @export
brainflow_python <- NULL
#' @export
np <- NULL
#' @export
pandas <- NULL
sys <- NULL
type_map <- NULL

.onLoad <- function(libname, pkgname)
{
    brainflow_python <<- import('brainflow', delay_load = TRUE)
    np <<- import('numpy', delay_load = TRUE)
    pandas <<- import('pandas', delay_load = TRUE)
    sys <<- import('sys', delay_load = TRUE)
    type_map <<- function(type)
    {
        if (is.character(type))
        {
            return (list(
                'float32' = np$float32,
                'float64' = np$float64,
                'auto' = NULL
            )[[type]])
        }
        type
    }
}
```

3.6 Matlab API Reference

Matlab binding calls C/C++ code as any other binding, it's not compatible with Octave.

A few general rules to keep in mind:

- Use char arrays instead strings to work with BrainFlow API, it means `'my_string'` instead `"my_string"`, otherwise you will get calllib error
- Use int32 values intead enums, it means `int32 (BoardIDs.SYNTHETIC_BOARD)` instead `BoardIDs.SYNTHETIC_BOARD`, the same is true for all enums in BrainFlow API

class brainflow.**AggOperations**
Bases: int32

Store all supported Agg Operations

class brainflow.**BoardIDs**
Bases: int32

Store all supported board ids

class brainflow.**BoardShim**(*board_id*, *input_params*)
BoardShim object to communicate with device

BoardShim constructor

board_id = None

static load_lib()

static check_ec(*ec*, *task_name*)

static set_log_level(*log_level*)
set log level for BoardShim

static set_log_file(*log_file*)
set log file for BoardShim, logger uses stderr by default

static enable_board_logger()
enable logger with level INFO

static enable_dev_board_logger()
enable logger with level TRACE

static disable_board_logger()
disable logger

static log_message(*log_level*, *message*)
write message to BoardShim logger

static get_sampling_rate(*board_id*)
get sampling rate for provided board id

static get_package_num_channel(*board_id*)
get package num channel for provided board id

static get_marker_channel(*board_id*)
get marker channel for provided board id

static get_battery_channel(*board_id*)
get battery channel for provided board id

static get_num_rows(*board_id*)
get num rows for provided board id

static get_timestamp_channel(*board_id*)
get timestamp channel for provided board id

static get_board_descr(*board_id*)
get board descr for provided board id

static get_eeg_names(*board_id*)
get eeg names for provided board id

static get_device_name(*board_id*)
get device name for provided board id

static get_eeg_channels(*board_id*)
get eeg channels for provided board id

static get_exg_channels(*board_id*)
get exg channels for provided board id

static get_emg_channels(*board_id*)
get emg channels for provided board id

static get_ecg_channels(*board_id*)
get ecg channels for provided board id

static get_eog_channels(*board_id*)
get eog channels for provided board id

static get_ppg_channels(*board_id*)
get ppg channels for provided board id

static get_eda_channels(*board_id*)
get eda channels for provided board id

static get_accel_channels(*board_id*)
get accel channels for provided board id

static get_analog_channels(*board_id*)
get analog channels for provided board id

static get_other_channels(*board_id*)
get other channels for provided board id

static get_temperature_channels(*board_id*)
get temperature channels for provided board id

static get_resistance_channels(*board_id*)
get resistance channels for provided board id

BoardShim(*board_id*, *input_params*)
BoardShim constructor

prepare_session()
prepare BoardShim session

config_board(*config*)
send string to the board

start_stream(*buffer_size*, *streamer_params*)
start data acqusition

stop_stream()
stop acqusition

release_session()
release session

insert_marker(*value*)
insert marker

get_board_data_count()
get amount of datapoints in internal buffer

get_board_data()
get all collected data and remove it from the buffer

get_current_board_data(*num_samples*)
get latest datapoints, doesnt remove it from internal buffer

is_prepared()
check if brainflow session prepared

class brainflow.**BrainFlowClassifiers**
Bases: int32

Store supported classifiers

class brainflow.**BrainFlowInputParams**
BrainFlow input params, check docs for params for your device

serial_port = None

BrainFlowInputParams()

to_json()

class brainflow.**BrainFlowMetrics**
Bases: int32

Store all supported metrics

class brainflow.**BrainFlowModelParams**(*metric*, *classifier*)
Store MLModel params

metric = None

BrainFlowModelParams(*metric*, *classifier*)

to_json()

class brainflow.**DataFilter**
DataFilter class for signal processing

static load_lib()

static check_ec(*ec*, *task_name*)

static set_log_level(*log_level*)
set log level for DataFilter

static set_log_file(*log_file*)
set log file for DataFilter

static enable_data_logger()
enable logger with level INFO

static enable_dev_data_logger()
enable logger with level TRACE

static disable_data_logger()
disable logger

static perform_lowpass(*data*, *sampling_rate*, *cutoff*, *order*, *filter_type*, *ripple*)
perform lowpass filtering

static perform_highpass(*data*, *sampling_rate*, *cutoff*, *order*, *filter_type*, *ripple*)
perform highpass filtering

static perform_bandpass(*data*, *sampling_rate*, *center_freq*, *band_width*, *order*, *filter_type*, *ripple*)
perform bandpass filtering

You need to provide center freqs and bandwidth

static perform_bandstop(*data*, *sampling_rate*, *center_freq*, *band_width*, *order*, *filter_type*, *ripple*)
perform bandpass filtering

You need to provide center freqs and bandwidth

static remove_environmental_noise(*data*, *sampling_rate*, *noise_type*)
perform noth filtering

static perform_rolling_filter(*data*, *period*, *operation*)
apply rolling filter

static detrend(*data*, *operation*)
remove trend from data

static perform_downsampling(*data*, *period*, *operation*)
downsample data

static perform_wavelet_transform(*data*, *wavelet*, *decomposition_level*)
perform wavelet transform

static perform_inverse_wavelet_transform(*wavelet_data*, *wavelet_sizes*, *original_data_len*, *wavelet*, *decomposition_level*)
perform inverse wavelet transform

static perform_wavelet_denoising(*data*, *wavelet*, *decomposition_level*)
perform wavelet denoising

static get_csp(*data*, *labels*)
get common spatial patterns

static get_window(*window_function*, *window_len*)
get window

static perform_fft(*data*, *window*)
perform fft

static perform_ifft(*fft_data*)
perform inverse fft

static get_avg_band_powers(*data*, *channels*, *sampling_rate*, *apply_filters*)
calculate average band powers

static get_psd(*data*, *sampling_rate*, *window*)
calculate PSD

static get_psd_welch(*data*, *nfft*, *overlap*, *sampling_rate*, *window*)
calculate PSD using welch method

static get_band_power(*ampls*, *freqs*, *freq_start*, *freq_end*)
calculate band power

static get_nearest_power_of_two(*value*)
get nearest power of two

static write_file(*data*, *file_name*, *file_mode*)
write data to file, in file data will be transposed

static read_file(*file_name*)
read data from file

class brainflow.**DetrendOperations**
Bases: int32

Store possible detrend operations

class brainflow.**ExitCodes**
Bases: int32

Store all possible exit codes

class brainflow.**FilterTypes**
Bases: int32

Store all possible filters

class brainflow.**IpProtocolType**
Bases: int32

Store all possible IP protocols

class brainflow.**LogLevels**
Bases: int32

Store all possible log levels

class brainflow.**MLModel**(*params*)
MLModel for inference

input_json = None

static load_lib()

static check_ec(*ec*, *task_name*)

static set_log_level(*log_level*)
set log level for MLModel

static set_log_file(*log_file*)
set log file for MLModel

static enable_ml_logger()
enable logger with level INFO

static enable_dev_ml_logger()
enable logger with level TRACE

static disable_ml_logger()
disable logger

MLModel(*params*)

prepare()
prepare model

release()
release model

predict(*input_data*)
perform inference for input data

class brainflow.**NoiseTypes**
Bases: int32

Store noise types

class brainflow.**WindowFunctions**
Bases: int32

Store window functions

3.7 Julia API Reference

Julia binding calls CC++ code as any other binding. Use Julia examples and API reference for other languaes as a starting point.

Since Julia is not Object-Oriented language, there is no DataFilter class. BoardShim class exists but all BoardShim class methods were moved to BrainFlow package and you need to pass BoardShim object to them.

Like here:

```
using BrainFlow


# specify logging library to use
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(5)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_current_board_data(32, board_shim)
BrainFlow.release_session(board_shim)

BrainFlow.write_file(data, "test.csv", "w")
restored_data = BrainFlow.read_file("test.csv")

println("Original Data")
println(data)
println("Restored Data")
println(restored_data)
```

CHAPTER

FOUR

DATA FORMAT DESCRIPTION

4.1 Units of Measure

For EEG, EMG, etc BrainFlow returns uV.

For timestamps BrainFlow uses UNIX timestamp, this count starts at the Unix Epoch on January 1st, 1970 at UTC. Precision is microsecond, but for some boards timestamps are generated on PC side as soon as package was received.

You can compare BrainFlow's timestamp with time returned by code like this:

```
import time
print (time.time ())
```

4.2 Generic Format Description

Methods like:

```
get_board_data ()
get_current_board_data (max_num_packages)
```

Return 2d double array [num_channels x num_data_points], rows of this array represent different channels like EEG channels, EMG channels, Accel channels, Timesteps and so on, while columns in this array represent actual packages from a board.

Exact format for this array is board specific. To keep the API uniform, we have methods like:

```
# these methods return an array of rows in this 2d array containing eeg\emg\ecg\accel␣
↪data
get_eeg_channels (board_id)
get_emg_channels (board_id)
get_ecg_channels (board_id)
get_accel_channels (board_id)
# and so on, check docs for full list
# also we have methods to get sampling rate from board id, get number of timestamp␣
↪channel and others
get_sampling_rate (board_id)
get_timestamp_channel (board_id)
# and so on
```

For some boards like OpenBCI Cyton, OpenBCI Ganglion, etc we cannot separate EMG, EEG, EDA and ECG and in this case we return exactly the same array for all these methods but for some devices EMG and EEG channels will differ.

If board has no such data these methods throw an exception with UNSUPPORTED_BOARD_ERROR exit code.

Using the methods above, you can write completely board agnostic code and switch boards using a single parameter! Even if you have only one board using these methods you can easily switch to Synthetic Board or Streaming Board.

4.3 OpenBCI Specific Data

4.3.1 Special Channels for OpenBCI Cyton Based Boards

Cyton-based boards from OpenBCI support different output formats, described here.

For Cyton based boards, we add Cyton End byte to a first channel from:

```
get_other_channels (board_id)
```

If Cyton End byte is equal to 0xC0 we add accel data. To get rows which contain accel data use:

```
get_accel_channels (board_id)
```

If Cyton End byte is equal to 0xC1 we add analog data. To get rows which contain analog data use:

```
get_analog_channels (board_id)
```

For analog data, we return int32 values. But since from low level API, we return double array, these values are converted to double without any changes.

Also we add raw unprocessed bytes to the second and next channels returned by:

```
get_other_channels (board_id)
```

If Cyton End Byte is outside this range, we drop the entire package.

Check this example for details:

```
import argparse
import time
import numpy as np

import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes


def main():
    parser = argparse.ArgumentParser()
    # use docs to check which parameters are required for specific board, e.g. for␣
↪Cyton - set serial port
    parser.add_argument('--ip-port', type=int, help='ip port', required=False,␣
↪default=0)
    parser.add_argument('--ip-protocol', type=int, help='ip protocol, check␣
↪IpProtocolType enum', required=False,
                        default=0)
    parser.add_argument('--ip-address', type=str, help='ip address', required=False,␣
↪default='')
    parser.add_argument('--serial-port', type=str, help='serial port', required=False,
↪ default='')
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('--mac-address', type=str, help='mac address', required=False,
↪ default='')
    parser.add_argument('--other-info', type=str, help='other info', required=False,␣
↪default='')
    parser.add_argument('--streamer-params', type=str, help='other info',␣
↪required=False, default='')
    parser.add_argument('--board-id', type=int, help='board id, check docs to get a␣
↪list of supported boards',
                        required=True)
    parser.add_argument('--log', action='store_true')
    args = parser.parse_args()

    params = BrainFlowInputParams()
    params.ip_port = args.ip_port
    params.serial_port = args.serial_port
    params.mac_address = args.mac_address
    params.other_info = args.other_info
    params.ip_address = args.ip_address
    params.ip_protocol = args.ip_protocol

    if (args.log):
        BoardShim.enable_dev_board_logger()
    else:
        BoardShim.disable_board_logger()

    board = BoardShim(args.board_id, params)
    board.prepare_session()

    board.start_stream()
    time.sleep(5)
    board.config_board('/2')  # enable analog mode only for Cyton Based Boards!
    time.sleep(5)
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    """
    data[BoardShim.get_other_channels(args.board_id)[0]] contains cyton end byte
    data[BoardShim.get_other_channels(args.board_id)[1....]] contains unprocessed␣
↪bytes
    if end byte is 0xC0 there are accel data in data[BoardShim.get_accel_
↪channels(args.board_id)[....]] else there are zeros
    if end byte is 0xC1 there are analog data in data[BoardShim.get_analog_
↪channels(args.board_id)[....]] else there are zeros
    """
    print(data[BoardShim.get_other_channels(args.board_id)[0]][0:5])  # should be␣
↪standard end byte 0xC0
    print(data[BoardShim.get_other_channels(args.board_id)[0]][-5:])  # should be␣
↪analog and byte 0xC1

    DataFilter.write_file(data, 'cyton_data.csv', 'w')


if __name__ == "__main__":
    main()
```

CHAPTER

FIVE

CODE SAMPLES

Make sure that you've installed BrainFlow package before running the code samples below.

See *Installation Instructions* for details.

5.1 Python

To run some signal processing samples, you may need to install:

- matplotlib
- pandas
- mne
- pyqtgraph

BrainFlow doesn't use these packages and doesn't install them, but the packages will be used in demos below.

5.1.1 Python Get Data from a Board

```
import argparse
import time
import numpy as np

import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations


def main():
    BoardShim.enable_dev_board_logger()

    parser = argparse.ArgumentParser()
    # use docs to check which parameters are required for specific board, e.g. for␣
↪Cyton - set serial port
    parser.add_argument('--timeout', type=int, help='timeout for device discovery or␣
↪connection', required=False,
                        default=0)
    parser.add_argument('--ip-port', type=int, help='ip port', required=False,␣
↪default=0)
    parser.add_argument('--ip-protocol', type=int, help='ip protocol, check␣
↪IpProtocolType enum', required=False,
```

(continues on next page)

(continued from previous page)

```
                        default=0)
    parser.add_argument('--ip-address', type=str, help='ip address', required=False,␣
↪default='')
    parser.add_argument('--serial-port', type=str, help='serial port', required=False,
↪ default='')
    parser.add_argument('--mac-address', type=str, help='mac address', required=False,
↪ default='')
    parser.add_argument('--other-info', type=str, help='other info', required=False,␣
↪default='')
    parser.add_argument('--streamer-params', type=str, help='streamer params',␣
↪required=False, default='')
    parser.add_argument('--serial-number', type=str, help='serial number',␣
↪required=False, default='')
    parser.add_argument('--board-id', type=int, help='board id, check docs to get a␣
↪list of supported boards',
                        required=True)
    parser.add_argument('--file', type=str, help='file', required=False, default='')
    args = parser.parse_args()

    params = BrainFlowInputParams()
    params.ip_port = args.ip_port
    params.serial_port = args.serial_port
    params.mac_address = args.mac_address
    params.other_info = args.other_info
    params.serial_number = args.serial_number
    params.ip_address = args.ip_address
    params.ip_protocol = args.ip_protocol
    params.timeout = args.timeout
    params.file = args.file

    board = BoardShim(args.board_id, params)
    board.prepare_session()

    # board.start_stream () # use this for default options
    board.start_stream(45000, args.streamer_params)
    time.sleep(10)
    # data = board.get_current_board_data (256) # get latest 256 packages or less,␣
↪doesnt remove them from internal buffer
    data = board.get_board_data()  # get all data and remove it from internal buffer
    board.stop_stream()
    board.release_session()

    print(data)


if __name__ == "__main__":
    main()
```

5.1.2 Python Markers

```
import argparse
import time
import numpy as np

import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations


def main():
    BoardShim.enable_dev_board_logger()

    parser = argparse.ArgumentParser()
    # use docs to check which parameters are required for specific board, e.g. for␣
↪Cyton - set serial port
    parser.add_argument('--timeout', type=int, help='timeout for device discovery or␣
↪connection', required=False,
                        default=0)
    parser.add_argument('--ip-port', type=int, help='ip port', required=False,␣
↪default=0)
    parser.add_argument('--ip-protocol', type=int, help='ip protocol, check␣
↪IpProtocolType enum', required=False,
                        default=0)
    parser.add_argument('--ip-address', type=str, help='ip address', required=False,␣
↪default='')
    parser.add_argument('--serial-port', type=str, help='serial port', required=False,
↪ default='')
    parser.add_argument('--mac-address', type=str, help='mac address', required=False,
↪ default='')
    parser.add_argument('--other-info', type=str, help='other info', required=False,␣
↪default='')
    parser.add_argument('--streamer-params', type=str, help='streamer params',␣
↪required=False, default='')
    parser.add_argument('--serial-number', type=str, help='serial number',␣
↪required=False, default='')
    parser.add_argument('--board-id', type=int, help='board id, check docs to get a␣
↪list of supported boards',
                        required=True)
    parser.add_argument('--file', type=str, help='file', required=False, default='')
    args = parser.parse_args()

    params = BrainFlowInputParams()
    params.ip_port = args.ip_port
    params.serial_port = args.serial_port
    params.mac_address = args.mac_address
    params.other_info = args.other_info
    params.serial_number = args.serial_number
    params.ip_address = args.ip_address
    params.ip_protocol = args.ip_protocol
    params.timeout = args.timeout
    params.file = args.file

    board = BoardShim(args.board_id, params)
    board.prepare_session()
```

(continues on next page)

(continued from previous page)

```
    board.start_stream(45000, args.streamer_params)
    for i in range(10):
        time.sleep(1)
        board.insert_marker(i + 1)
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    print(data)


if __name__ == "__main__":
    main()
```

5.1.3 Python Read Write File

```
import argparse
import time
import numpy as np
import pandas as pd

import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations


def main():
    BoardShim.enable_dev_board_logger()

    # use synthetic board for demo
    params = BrainFlowInputParams()
    board = BoardShim(BoardIds.SYNTHETIC_BOARD.value, params)
    board.prepare_session()
    board.start_stream()
    BoardShim.log_message(LogLevels.LEVEL_INFO.value, 'start sleeping in the main␣
↪thread')
    time.sleep(10)
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    # demo how to convert it to pandas DF and plot data
    eeg_channels = BoardShim.get_eeg_channels(BoardIds.SYNTHETIC_BOARD.value)
    df = pd.DataFrame(np.transpose(data))
    print('Data From the Board')
    print(df.head(10))

    # demo for data serialization using brainflow API, we recommend to use it instead␣
↪pandas.to_csv()
    DataFilter.write_file(data, 'test.csv', 'w')  # use 'a' for append mode
    restored_data = DataFilter.read_file('test.csv')
    restored_df = pd.DataFrame(np.transpose(restored_data))
    print('Data From the File')
    print(restored_df.head(10))
```

(continues on next page)

(continued from previous page)

```
if __name__ == "__main__":
    main()
```

5.1.4 Python Downsample Data

```
import time
import numpy as np

import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations


def main():
    BoardShim.enable_dev_board_logger()

    # use synthetic board for demo
    params = BrainFlowInputParams()
    board = BoardShim(BoardIds.SYNTHETIC_BOARD.value, params)
    board.prepare_session()
    board.start_stream()
    BoardShim.log_message(LogLevels.LEVEL_INFO.value, 'start sleeping in the main␣
↪thread')
    time.sleep(10)
    data = board.get_current_board_data(20)  # get 20 latest data points dont remove␣
↪them from internal buffer
    board.stop_stream()
    board.release_session()

    eeg_channels = BoardShim.get_eeg_channels(BoardIds.SYNTHETIC_BOARD.value)
    # demo for downsampling, it just aggregates data
    for count, channel in enumerate(eeg_channels):
        print('Original data for channel %d:' % channel)
        print(data[channel])
        if count == 0:
            downsampled_data = DataFilter.perform_downsampling(data[channel], 3,␣
↪AggOperations.MEDIAN.value)
        elif count == 1:
            downsampled_data = DataFilter.perform_downsampling(data[channel], 2,␣
↪AggOperations.MEAN.value)
        else:
            downsampled_data = DataFilter.perform_downsampling(data[channel], 2,␣
↪AggOperations.EACH.value)
        print('Downsampled data for channel %d:' % channel)
        print(downsampled_data)


if __name__ == "__main__":
    main()
```

5.1.5 Python Transforms

```
import argparse
import time
import brainflow
import numpy as np

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations,␣
↪WindowFunctions


def main():
    BoardShim.enable_dev_board_logger()

    # use synthetic board for demo
    params = BrainFlowInputParams()
    board_id = BoardIds.SYNTHETIC_BOARD.value
    sampling_rate = BoardShim.get_sampling_rate(board_id)
    board = BoardShim(board_id, params)
    board.prepare_session()
    board.start_stream()
    BoardShim.log_message(LogLevels.LEVEL_INFO.value, 'start sleeping in the main␣
↪thread')
    time.sleep(10)
    data = board.get_current_board_data(DataFilter.get_nearest_power_of_two(sampling_
↪rate))
    board.stop_stream()
    board.release_session()

    eeg_channels = BoardShim.get_eeg_channels(board_id)
    # demo for transforms
    for count, channel in enumerate(eeg_channels):
        print('Original data for channel %d:' % channel)
        print(data[channel])
        # demo for wavelet transforms
        # wavelet_coeffs format is[A(J) D(J) D(J-1) ..... D(1)] where J is␣
↪decomposition level, A - app coeffs, D - detailed coeffs
        # lengths array stores lengths for each block
        wavelet_coeffs, lengths = DataFilter.perform_wavelet_transform(data[channel],
↪'db5', 3)
        app_coefs = wavelet_coeffs[0: lengths[0]]
        detailed_coeffs_first_block = wavelet_coeffs[lengths[0]: lengths[1]]
        # you can do smth with wavelet coeffs here, for example denoising works via␣
↪thresholds
        # for wavelets coefficients
        restored_data = DataFilter.perform_inverse_wavelet_transform((wavelet_coeffs,␣
↪lengths), data[channel].shape[0],
                                                                     'db5', 3)
        print('Restored data after wavelet transform for channel %d:' % channel)
        print(restored_data)

        # demo for fft, len of data must be a power of 2
        fft_data = DataFilter.perform_fft(data[channel], WindowFunctions.NO_WINDOW.
↪value)
        # len of fft_data is N / 2 + 1
        restored_fft_data = DataFilter.perform_ifft(fft_data)
```

(continues on next page)

(continued from previous page)

```
        print('Restored data after fft for channel %d:' % channel)
        print(restored_fft_data)


if __name__ == "__main__":
    main()
```

5.1.6 Python Signal Filtering

```
import argparse
import time
import brainflow
import numpy as np

import pandas as pd
import matplotlib

matplotlib.use('Agg')
import matplotlib.pyplot as plt

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations, NoiseTypes


def main():
    BoardShim.enable_dev_board_logger()

    # use synthetic board for demo
    params = BrainFlowInputParams()
    board_id = BoardIds.SYNTHETIC_BOARD.value
    board = BoardShim(board_id, params)
    board.prepare_session()
    board.start_stream()
    BoardShim.log_message(LogLevels.LEVEL_INFO.value, 'start sleeping in the main␣
↪thread')
    time.sleep(10)
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    # demo how to convert it to pandas DF and plot data
    eeg_channels = BoardShim.get_eeg_channels(board_id)
    df = pd.DataFrame(np.transpose(data))
    plt.figure()
    df[eeg_channels].plot(subplots=True)
    plt.savefig('before_processing.png')

    # for demo apply different filters to different channels, in production choose one
    for count, channel in enumerate(eeg_channels):
        # filters work in-place
        if count == 0:
            DataFilter.perform_bandpass(data[channel], BoardShim.get_sampling_
↪rate(board_id), 15.0, 6.0, 4,
                                        FilterTypes.BESSEL.value, 0)
        elif count == 1:
```

(continues on next page)

(continued from previous page)

```
            DataFilter.perform_bandstop(data[channel], BoardShim.get_sampling_
↪rate(board_id), 30.0, 1.0, 3,
                                        FilterTypes.BUTTERWORTH.value, 0)
        elif count == 2:
            DataFilter.perform_lowpass(data[channel], BoardShim.get_sampling_
↪rate(board_id), 20.0, 5,
                                       FilterTypes.CHEBYSHEV_TYPE_1.value, 1)
        elif count == 3:
            DataFilter.perform_highpass(data[channel], BoardShim.get_sampling_
↪rate(board_id), 3.0, 4,
                                        FilterTypes.BUTTERWORTH.value, 0)
        elif count == 4:
            DataFilter.perform_rolling_filter(data[channel], 3, AggOperations.MEAN.
↪value)
        else:
            DataFilter.remove_environmental_noise(data[channel], BoardShim.get_
↪sampling_rate(board_id), NoiseTypes.FIFTY.value)

    df = pd.DataFrame(np.transpose(data))
    plt.figure()
    df[eeg_channels].plot(subplots=True)
    plt.savefig('after_processing.png')


if __name__ == "__main__":
    main()
```

5.1.7 Python Denoising

```
import argparse
import time
import brainflow
import numpy as np

import pandas as pd
import matplotlib

matplotlib.use('Agg')
import matplotlib.pyplot as plt

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations


def main():
    BoardShim.enable_dev_board_logger()

    # use synthetic board for demo
    params = BrainFlowInputParams()
    board_id = BoardIds.SYNTHETIC_BOARD.value
    board = BoardShim(board_id, params)
    board.prepare_session()
    board.start_stream()
    BoardShim.log_message(LogLevels.LEVEL_INFO.value, 'start sleeping in the main␣
↪thread')
```

(continues on next page)

(continued from previous page)

```
    time.sleep(20)
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    # demo how to convert it to pandas DF and plot data
    eeg_channels = BoardShim.get_eeg_channels(board_id)
    df = pd.DataFrame(np.transpose(data))
    plt.figure()
    df[eeg_channels].plot(subplots=True)
    plt.savefig('before_processing.png')

    # demo for denoising, apply different methods to different channels for demo
    for count, channel in enumerate(eeg_channels):
        # first of all you can try simple moving median or moving average with␣
↪different window size
        if count == 0:
            DataFilter.perform_rolling_filter(data[channel], 3, AggOperations.MEAN.
↪value)
        elif count == 1:
            DataFilter.perform_rolling_filter(data[channel], 3, AggOperations.MEDIAN.
↪value)
        # if methods above dont work for your signal you can try wavelet based␣
↪denoising
        # feel free to try different functions and decomposition levels
        elif count == 2:
            DataFilter.perform_wavelet_denoising(data[channel], 'db6', 3)
        elif count == 3:
            DataFilter.perform_wavelet_denoising(data[channel], 'bior3.9', 3)
        elif count == 4:
            DataFilter.perform_wavelet_denoising(data[channel], 'sym7', 3)
        elif count == 5:
            # with synthetic board this one looks like the best option, but it␣
↪depends on many circumstances
            DataFilter.perform_wavelet_denoising(data[channel], 'coif3', 3)

    df = pd.DataFrame(np.transpose(data))
    plt.figure()
    df[eeg_channels].plot(subplots=True)
    plt.savefig('after_processing.png')


if __name__ == "__main__":
    main()
```

5.1.8 Python MNE Integration

```
import time
import numpy as np
import matplotlib

matplotlib.use('Agg')
import matplotlib.pyplot as plt
import pandas as pd
```

(continues on next page)

(continued from previous page)

```
import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds

import mne
from mne.channels import read_layout


def main():
    BoardShim.enable_dev_board_logger()
    # use synthetic board for demo
    params = BrainFlowInputParams()
    board = BoardShim(BoardIds.SYNTHETIC_BOARD.value, params)
    board.prepare_session()
    board.start_stream()
    time.sleep(10)
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    eeg_channels = BoardShim.get_eeg_channels(BoardIds.SYNTHETIC_BOARD.value)
    eeg_data = data[eeg_channels, :]
    eeg_data = eeg_data / 1000000  # BrainFlow returns uV, convert to V for MNE

    # Creating MNE objects from brainflow data arrays
    ch_types = ['eeg'] * len(eeg_channels)
    ch_names = BoardShim.get_eeg_names(BoardIds.SYNTHETIC_BOARD.value)
    sfreq = BoardShim.get_sampling_rate(BoardIds.SYNTHETIC_BOARD.value)
    info = mne.create_info(ch_names=ch_names, sfreq=sfreq, ch_types=ch_types)
    raw = mne.io.RawArray(eeg_data, info)
    # its time to plot something!
    raw.plot_psd(average=True)
    plt.savefig('psd.png')


if __name__ == '__main__':
    main()
```

5.1.9 Python Band Power

```
import argparse
import time
import brainflow
import numpy as np

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations,␣
↪WindowFunctions, DetrendOperations


def main():
    BoardShim.enable_dev_board_logger()

    # use synthetic board for demo
    params = BrainFlowInputParams()
    board_id = BoardIds.SYNTHETIC_BOARD.value
```

(continues on next page)

(continued from previous page)

```
    board_descr = BoardShim.get_board_descr(board_id)
    sampling_rate = int(board_descr['sampling_rate'])
    board = BoardShim(board_id, params)
    board.prepare_session()
    board.start_stream()
    BoardShim.log_message(LogLevels.LEVEL_INFO.value, 'start sleeping in the main␣
↪thread')
    time.sleep(10)
    nfft = DataFilter.get_nearest_power_of_two(sampling_rate)
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    eeg_channels = board_descr['eeg_channels']
    # second eeg channel of synthetic board is a sine wave at 10Hz, should see huge␣
↪alpha
    eeg_channel = eeg_channels[1]
    # optional detrend
    DataFilter.detrend(data[eeg_channel], DetrendOperations.LINEAR.value)
    psd = DataFilter.get_psd_welch(data[eeg_channel], nfft, nfft // 2, sampling_rate,
                                   WindowFunctions.BLACKMAN_HARRIS.value)

    band_power_alpha = DataFilter.get_band_power(psd, 7.0, 13.0)
    band_power_beta = DataFilter.get_band_power(psd, 14.0, 30.0)
    print("alpha/beta:%f", band_power_alpha / band_power_beta)

    # fail test if ratio is not smth we expect
    if (band_power_alpha / band_power_beta < 100):
        raise ValueError('Wrong Ratio')


if __name__ == "__main__":
    main()
```

5.1.10 Python EEG Metrics

```
import argparse
import time
import brainflow
import numpy as np

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds,
↪ BrainFlowError
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations,␣
↪WindowFunctions, DetrendOperations
from brainflow.ml_model import MLModel, BrainFlowMetrics, BrainFlowClassifiers,␣
↪BrainFlowModelParams
from brainflow.exit_codes import *


def main():
    BoardShim.enable_board_logger()
    DataFilter.enable_data_logger()
    MLModel.enable_ml_logger()
```

(continues on next page)

(continued from previous page)

```
    parser = argparse.ArgumentParser()
    # use docs to check which parameters are required for specific board, e.g. for␣
↪Cyton - set serial port
    parser.add_argument('--timeout', type=int, help='timeout for device discovery or␣
↪connection', required=False,
                        default=0)
    parser.add_argument('--ip-port', type=int, help='ip port', required=False,␣
↪default=0)
    parser.add_argument('--ip-protocol', type=int, help='ip protocol, check␣
↪IpProtocolType enum', required=False,
                        default=0)
    parser.add_argument('--ip-address', type=str, help='ip address', required=False,␣
↪default='')
    parser.add_argument('--serial-port', type=str, help='serial port', required=False,
↪ default='')
    parser.add_argument('--mac-address', type=str, help='mac address', required=False,
↪ default='')
    parser.add_argument('--other-info', type=str, help='other info', required=False,␣
↪default='')
    parser.add_argument('--streamer-params', type=str, help='streamer params',␣
↪required=False, default='')
    parser.add_argument('--serial-number', type=str, help='serial number',␣
↪required=False, default='')
    parser.add_argument('--board-id', type=int, help='board id, check docs to get a␣
↪list of supported boards',
                        required=True)
    parser.add_argument('--file', type=str, help='file', required=False, default='')
    args = parser.parse_args()

    params = BrainFlowInputParams()
    params.ip_port = args.ip_port
    params.serial_port = args.serial_port
    params.mac_address = args.mac_address
    params.other_info = args.other_info
    params.serial_number = args.serial_number
    params.ip_address = args.ip_address
    params.ip_protocol = args.ip_protocol
    params.timeout = args.timeout
    params.file = args.file

    board = BoardShim(args.board_id, params)
    master_board_id = board.get_board_id()
    sampling_rate = BoardShim.get_sampling_rate(master_board_id)
    board.prepare_session()
    board.start_stream(45000, args.streamer_params)
    BoardShim.log_message(LogLevels.LEVEL_INFO.value, 'start sleeping in the main␣
↪thread')
    time.sleep(5)  # recommended window size for eeg metric calculation is at least 4␣
↪seconds, bigger is better
    data = board.get_board_data()
    board.stop_stream()
    board.release_session()

    eeg_channels = BoardShim.get_eeg_channels(int(master_board_id))
    bands = DataFilter.get_avg_band_powers(data, eeg_channels, sampling_rate, True)
    feature_vector = np.concatenate((bands[0], bands[1]))
    print(feature_vector)
```

(continues on next page)

(continued from previous page)

```
    # calc concentration
    concentration_params = BrainFlowModelParams(BrainFlowMetrics.CONCENTRATION.value,␣
↪BrainFlowClassifiers.KNN.value)
    concentration = MLModel(concentration_params)
    concentration.prepare()
    print('Concentration: %f' % concentration.predict(feature_vector))
    concentration.release()

    # calc relaxation
    relaxation_params = BrainFlowModelParams(BrainFlowMetrics.RELAXATION.value,␣
↪BrainFlowClassifiers.REGRESSION.value)
    relaxation = MLModel(relaxation_params)
    relaxation.prepare()
    print('Relaxation: %f' % relaxation.predict(feature_vector))
    relaxation.release()


if __name__ == "__main__":
    main()
```

5.1.11 Python Real Time Plot

```
import argparse
import time
import logging
import random

import pyqtgraph as pg
from pyqtgraph.Qt import QtGui, QtCore

import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds,␣
↪BrainFlowError
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations,␣
↪WindowFunctions, DetrendOperations


class Graph:
    def __init__(self, board_shim):
        pg.setConfigOption('background', 'w')
        pg.setConfigOption('foreground', 'k')

        self.board_id = board_shim.get_board_id()
        self.board_shim = board_shim
        self.exg_channels = BoardShim.get_exg_channels(self.board_id)
        self.sampling_rate = BoardShim.get_sampling_rate(self.board_id)
        self.update_speed_ms = 50
        self.window_size = 4
        self.num_points = self.window_size * self.sampling_rate

        self.app = QtGui.QApplication([])
        self.win = pg.GraphicsWindow(title='BrainFlow Plot',size=(800, 600))

        self._init_pens()
```

(continues on next page)

(continued from previous page)

```
        self._init_timeseries()
        self._init_psd()
        self._init_band_plot()

        timer = QtCore.QTimer()
        timer.timeout.connect(self.update)
        timer.start(self.update_speed_ms)
        QtGui.QApplication.instance().exec_()

    def _init_pens(self):
        self.pens = list()
        self.brushes = list()
        colors = ['#A54E4E', '#A473B6', '#5B45A4', '#2079D2', '#32B798', '#2FA537', '
↪#9DA52F', '#A57E2F', '#A53B2F']
        for i in range(len(colors)):
            pen = pg.mkPen({'color': colors[i], 'width': 2})
            self.pens.append(pen)
            brush = pg.mkBrush(colors[i])
            self.brushes.append(brush)

    def _init_timeseries(self):
        self.plots = list()
        self.curves = list()
        for i in range(len(self.exg_channels)):
            p = self.win.addPlot(row=i,col=0)
            p.showAxis('left', False)
            p.setMenuEnabled('left', False)
            p.showAxis('bottom', False)
            p.setMenuEnabled('bottom', False)
            if i == 0:
                p.setTitle('TimeSeries Plot')
            self.plots.append(p)
            hist_pen = pg.mkPen((170, 57, 57, 255), width=1.)
            curve = p.plot(pen=self.pens[i % len(self.pens)])
            #curve.setDownsampling(auto=True, method='mean', ds=3)
            self.curves.append(curve)

    def _init_psd(self):
        self.psd_plot = self.win.addPlot(row=0,col=1, rowspan=len(self.exg_channels)//
↪2)
        self.psd_plot.showAxis('left', False)
        self.psd_plot.setMenuEnabled('left', False)
        self.psd_plot.setTitle('PSD Plot')
        self.psd_plot.setLogMode(False, True)
        self.psd_curves = list()
        self.psd_size = DataFilter.get_nearest_power_of_two(self.sampling_rate)
        for i in range(len(self.exg_channels)):
            psd_curve = self.psd_plot.plot(pen=self.pens[i % len(self.pens)])
            psd_curve.setDownsampling(auto=True, method='mean', ds=3)
            self.psd_curves.append(psd_curve)

    def _init_band_plot(self):
        self.band_plot = self.win.addPlot(row=len(self.exg_channels)//2, col=1,␣
↪rowspan=len(self.exg_channels)//2)
        self.band_plot.showAxis('left', False)
        self.band_plot.setMenuEnabled('left', False)
        self.band_plot.showAxis('bottom', False)
```

(continues on next page)

(continued from previous page)

```
        self.band_plot.setMenuEnabled('bottom', False)
        self.band_plot.setTitle('BandPower Plot')
        y = [0, 0, 0, 0, 0]
        x = [1, 2, 3, 4, 5]
        self.band_bar = pg.BarGraphItem(x=x, height=y, width=0.8, pen=self.pens[0],␣
↪brush=self.brushes[0])
        self.band_plot.addItem(self.band_bar)

    def update(self):
        data = self.board_shim.get_current_board_data(self.num_points)
        avg_bands = [0, 0, 0, 0, 0]
        for count, channel in enumerate(self.exg_channels):
            # plot timeseries
            DataFilter.detrend(data[channel], DetrendOperations.LINEAR.value)
            DataFilter.perform_bandpass(data[channel], self.sampling_rate, 30.0, 56.0,
↪ 2,
                                        FilterTypes.BUTTERWORTH.value, 0)
            DataFilter.perform_bandstop(data[channel], self.sampling_rate, 50.0, 4.0,␣
↪2,
                                        FilterTypes.BUTTERWORTH.value, 0)
            DataFilter.perform_bandstop(data[channel], self.sampling_rate, 60.0, 4.0,␣
↪2,
                                        FilterTypes.BUTTERWORTH.value, 0)
            self.curves[count].setData(data[channel].tolist())
            if data.shape[1] > self.psd_size:
                # plot psd
                psd_data = DataFilter.get_psd_welch(data[channel], self.psd_size,␣
↪self.psd_size // 2, self.sampling_rate,
                                    WindowFunctions.BLACKMAN_HARRIS.value)
                lim = min(70, len(psd_data[0]))
                self.psd_curves[count].setData(psd_data[1][0:lim].tolist(), psd_
↪data[0][0:lim].tolist())
                # plot bands
                avg_bands[0] = avg_bands[0] + DataFilter.get_band_power(psd_data, 1.0,
↪ 4.0)
                avg_bands[1] = avg_bands[1] + DataFilter.get_band_power(psd_data, 4.0,
↪ 8.0)
                avg_bands[2] = avg_bands[2] + DataFilter.get_band_power(psd_data, 8.0,
↪ 13.0)
                avg_bands[3] = avg_bands[3] + DataFilter.get_band_power(psd_data, 13.
↪0, 30.0)
                avg_bands[4] = avg_bands[4] + DataFilter.get_band_power(psd_data, 30.
↪0, 50.0)

        avg_bands = [int(x * 100 / len(self.exg_channels)) for x in avg_bands]
        self.band_bar.setOpts(height=avg_bands)

        self.app.processEvents()


def main():
    BoardShim.enable_dev_board_logger()
    logging.basicConfig(level=logging.DEBUG)

    parser = argparse.ArgumentParser()
    # use docs to check which parameters are required for specific board, e.g. for␣
↪Cyton - set serial port
```

(continues on next page)

(continued from previous page)

```
    parser.add_argument('--timeout', type=int, help='timeout for device discovery or␣
↪connection', required=False,
                        default=0)
    parser.add_argument('--ip-port', type=int, help='ip port', required=False,␣
↪default=0)
    parser.add_argument('--ip-protocol', type=int, help='ip protocol, check␣
↪IpProtocolType enum', required=False,
                        default=0)
    parser.add_argument('--ip-address', type=str, help='ip address', required=False,␣
↪default='')
    parser.add_argument('--serial-port', type=str, help='serial port', required=False,
↪ default='')
    parser.add_argument('--mac-address', type=str, help='mac address', required=False,
↪ default='')
    parser.add_argument('--other-info', type=str, help='other info', required=False,␣
↪default='')
    parser.add_argument('--streamer-params', type=str, help='streamer params',␣
↪required=False, default='')
    parser.add_argument('--serial-number', type=str, help='serial number',␣
↪required=False, default='')
    parser.add_argument('--board-id', type=int, help='board id, check docs to get a␣
↪list of supported boards',
                        required=False, default=BoardIds.SYNTHETIC_BOARD)
    parser.add_argument('--file', type=str, help='file', required=False, default='')
    args = parser.parse_args()

    params = BrainFlowInputParams()
    params.ip_port = args.ip_port
    params.serial_port = args.serial_port
    params.mac_address = args.mac_address
    params.other_info = args.other_info
    params.serial_number = args.serial_number
    params.ip_address = args.ip_address
    params.ip_protocol = args.ip_protocol
    params.timeout = args.timeout
    params.file = args.file

    try:
        board_shim = BoardShim(args.board_id, params)
        board_shim.prepare_session()
        board_shim.start_stream(450000, args.streamer_params)
        g = Graph(board_shim)
    except BaseException as e:
        logging.warning('Exception', exc_info=True)
    finally:
        if board_shim.is_prepared():
            logging.info('Releasing session')
            board_shim.release_session()


if __name__ == '__main__':
    main()
```

5.2 Java

5.2.1 Java Get Data from a Board

```
package brainflow.examples;

import java.util.Arrays;

import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.LogLevels;

public class BrainFlowGetData
{

    public static void main (String[] args) throws Exception
    {
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = parse_args (args, params);
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        // board_shim.start_stream (); // use this for default options
        board_shim.start_stream (450000, "file://file_stream.csv:w");
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in
↪the main thread");
        Thread.sleep (5000);
        board_shim.stop_stream ();
        System.out.println (board_shim.get_board_data_count ());
        double[][] data = board_shim.get_current_board_data (30); // doesnt flush it
↪from ring buffer
        // double[][] data = board_shim.get_board_data (); // get all data and flush
        // from ring buffer
        for (int i = 0; i < data.length; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
        board_shim.release_session ();
    }

    private static int parse_args (String[] args, BrainFlowInputParams params)
    {
        int board_id = -1;
        for (int i = 0; i < args.length; i++)
        {
            if (args[i].equals ("--ip-address"))
            {
                params.ip_address = args[i + 1];
            }
            if (args[i].equals ("--serial-port"))
            {
                params.serial_port = args[i + 1];
            }
            if (args[i].equals ("--ip-port"))
            {
```

(continues on next page)

(continued from previous page)

```
                params.ip_port = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--ip-protocol"))
            {
                params.ip_protocol = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--other-info"))
            {
                params.other_info = args[i + 1];
            }
            if (args[i].equals ("--board-id"))
            {
                board_id = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--timeout"))
            {
                params.timeout = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--serial-number"))
            {
                params.serial_number = args[i + 1];
            }
            if (args[i].equals ("--file"))
            {
                params.file = args[i + 1];
            }
        }
        return board_id;
    }
}
```

5.2.2 Java Markers

```
package brainflow.examples;

import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;

public class Markers
{

    public static void main (String[] args) throws Exception
    {
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = parse_args (args, params);
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (450000, "file://file_stream.csv:w");
        for (int i = 1; i < 5; i++)
        {
            Thread.sleep (1000);
            board_shim.insert_marker (i);
        }
```

(continues on next page)

(continued from previous page)

```
        board_shim.stop_stream ();
        board_shim.release_session ();
    }

    private static int parse_args (String[] args, BrainFlowInputParams params)
    {
        int board_id = -1;
        for (int i = 0; i < args.length; i++)
        {
            if (args[i].equals ("--ip-address"))
            {
                params.ip_address = args[i + 1];
            }
            if (args[i].equals ("--serial-port"))
            {
                params.serial_port = args[i + 1];
            }
            if (args[i].equals ("--ip-port"))
            {
                params.ip_port = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--ip-protocol"))
            {
                params.ip_protocol = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--other-info"))
            {
                params.other_info = args[i + 1];
            }
            if (args[i].equals ("--board-id"))
            {
                board_id = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--timeout"))
            {
                params.timeout = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--serial-number"))
            {
                params.serial_number = args[i + 1];
            }
            if (args[i].equals ("--file"))
            {
                params.file = args[i + 1];
            }
        }
        return board_id;
    }
}
```

5.2.3 Java Read Write File

```
package brainflow.examples;

import java.util.Arrays;

import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
import brainflow.LogLevels;

public class Serialization
{

    public static void main (String[] args) throws Exception
    {
        // use Synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = BoardIds.SYNTHETIC_BOARD.get_code ();
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in␣
↪the main thread");
        Thread.sleep (5000);
        board_shim.stop_stream ();
        System.out.println (board_shim.get_board_data_count ());
        int num_rows = BoardShim.get_num_rows (board_id);
        double[][] data = board_shim.get_current_board_data (30);
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
        board_shim.release_session ();

        // demo for serialization
        DataFilter.write_file (data, "test.csv", "w");
        double[][] restored_data = DataFilter.read_file ("test.csv");
        System.out.println ("After Serialization:");
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (restored_data[i]));
        }
    }

}
```

5.2.4 Java Downsample Data

```
package brainflow.examples;

import java.util.Arrays;

import brainflow.AggOperations;
import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
import brainflow.LogLevels;

public class Downsampling
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = BoardIds.SYNTHETIC_BOARD.get_code ();

        BoardShim board_shim = new BoardShim (board_id, params);
        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in␣
↪the main thread");
        Thread.sleep (5000);
        board_shim.stop_stream ();
        System.out.println (board_shim.get_board_data_count ());
        double[][] data = board_shim.get_current_board_data (30);
        board_shim.release_session ();

        int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
        for (int i = 0; i < eeg_channels.length; i++)
        {
            System.out.println ("Original data:");
            System.out.println (Arrays.toString (data[i]));
            // keep each second element, you can use MEAN and MEDIAN as well
            double[] downsampled_data = DataFilter.perform_downsampling (data[eeg_
↪channels[i]], 2,
                    AggOperations.EACH.get_code ());
            System.out.println ("Downsampled data:");
            System.out.println (Arrays.toString (downsampled_data));
        }
    }
}
```

5.2.5 Java Transforms

```
package brainflow.examples;

import java.util.Arrays;

import org.apache.commons.lang3.tuple.Pair;
import org.apache.commons.math3.complex.Complex;

import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
import brainflow.LogLevels;
import brainflow.WindowFunctions;

public class Transforms
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = BoardIds.SYNTHETIC_BOARD.get_code ();
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in␣
↪the main thread");
        Thread.sleep (10000);
        board_shim.stop_stream ();
        System.out.println (board_shim.get_board_data_count ());
        int num_rows = BoardShim.get_num_rows (board_id);
        double[][] data = board_shim.get_current_board_data (64);
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
        board_shim.release_session ();

        int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
        for (int i = 0; i < eeg_channels.length; i++)
        {
            System.out.println ("Original data:");
            System.out.println (Arrays.toString (data[eeg_channels[i]]));
            // demo for wavelet transform
            // Pair of coeffs array in format[A(J) D(J) D(J-1) ..... D(1)] where J is␣
↪a
            // decomposition level, A - app coeffs, D - detailed coeffs, and array␣
↪which
            // stores
            // length for each block, len of this array is decomposition_length + 1
            Pair<double[], int[]> wavelet_data = DataFilter.perform_wavelet_transform␣
↪(data[eeg_channels[i]], "db4", 3);
            // print approximation coeffs
```

(continues on next page)

(continued from previous page)

```
            for (int j = 0; j < wavelet_data.getRight ()[0]; j++)
            {
                System.out.print (wavelet_data.getLeft ()[j] + " ");
            }
            System.out.println ();
            // you can do smth with these coeffs here, for example denoising works via
            // thresholds for wavelet coeffs
            double[] restored_data = DataFilter.perform_inverse_wavelet_transform␣
↪(wavelet_data,
                    data[eeg_channels[i]].length, "db4", 3);
            System.out.println ("Restored data after wavelet:");
            System.out.println (Arrays.toString (restored_data));

            // demo for fft works only for power of 2
            // len of fft_data is N / 2 + 1
            Complex[] fft_data = DataFilter.perform_fft (data[eeg_channels[i]], 0, 64,
                    WindowFunctions.NO_WINDOW.get_code ());
            double[] restored_fft_data = DataFilter.perform_ifft (fft_data);
            System.out.println ("Restored data after fft:");
            System.out.println (Arrays.toString (restored_fft_data));
        }
    }
}
```

5.2.6 Java Signal Filtering

```
package brainflow.examples;

import java.util.Arrays;

import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
import brainflow.FilterTypes;
import brainflow.LogLevels;
import brainflow.NoiseTypes;

public class SignalFiltering
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = BoardIds.SYNTHETIC_BOARD.get_code ();
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in␣
↪the main thread");
        Thread.sleep (5000);
        board_shim.stop_stream ();
```

(continues on next page)

(continued from previous page)

```
        System.out.println (board_shim.get_board_data_count ());
        int num_rows = BoardShim.get_num_rows (board_id);
        double[][] data = board_shim.get_current_board_data (30);
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
        board_shim.release_session ();

        int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
        for (int i = 0; i < eeg_channels.length; i++)
        {
            // just for demo - apply different filters to different eeg channels
            switch (i)
            {
                case 0:
                    DataFilter.perform_lowpass (data[eeg_channels[i]], BoardShim.get_
↪sampling_rate (board_id), 20.0, 4,
                            FilterTypes.BESSEL.get_code (), 0.0);
                    break;
                case 1:
                    DataFilter.perform_highpass (data[eeg_channels[i]], BoardShim.get_
↪sampling_rate (board_id), 5.0, 4,
                            FilterTypes.BUTTERWORTH.get_code (), 0.0);
                    break;
                case 2:
                    DataFilter.perform_bandpass (data[eeg_channels[i]], BoardShim.get_
↪sampling_rate (board_id), 15.0,
                            5.0, 4, FilterTypes.CHEBYSHEV_TYPE_1.get_code (), 1.0);
                    break;
                case 3:
                    DataFilter.perform_bandstop (data[eeg_channels[i]], BoardShim.get_
↪sampling_rate (board_id), 50.0,
                            1.0, 4, FilterTypes.CHEBYSHEV_TYPE_1.get_code (), 1.0);
                    break;
                default:
                    DataFilter.remove_environmental_noise (data[eeg_channels[i]],
                            BoardShim.get_sampling_rate (board_id), NoiseTypes.FIFTY.
↪get_code ());;
                    break;
            }
        }
        System.out.println ("After signal processing:");
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
    }
}
```

5.2.7 Java Denoising

```
package brainflow.examples;

import brainflow.AggOperations;
import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
import brainflow.LogLevels;

import java.util.Arrays;

public class Denoising
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = BoardIds.SYNTHETIC_BOARD.get_code ();
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in␣
↪the main thread");
        Thread.sleep (5000);
        board_shim.stop_stream ();
        System.out.println (board_shim.get_board_data_count ());
        int num_rows = BoardShim.get_num_rows (board_id);
        double[][] data = board_shim.get_current_board_data (64);
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
        board_shim.release_session ();

        int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
        for (int i = 0; i < eeg_channels.length; i++)
        {
            // just for demo - apply different methods to different eeg channels
            switch (i)
            {
                // first of all you can try simple moving average or moving median
                case 0:
                    DataFilter.perform_rolling_filter (data[eeg_channels[i]], 3,␣
↪AggOperations.MEAN.get_code ());
                    break;
                case 1:
                    DataFilter.perform_rolling_filter (data[eeg_channels[i]], 3,␣
↪AggOperations.MEDIAN.get_code ());
                    break;
                // if methods above dont work good for you you should try wavelet␣
↪based
                // denoising
```

(continues on next page)

(continued from previous page)

```
                default:
                    // try different functions and different decomposition levels here
                    DataFilter.perform_wavelet_denoising (data[eeg_channels[i]], "db4
↪", 3);
                    break;
            }
        }
        System.out.println ("After signal processing:");
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
    }
}
```

5.2.8 Java Band Power

```
package brainflow.examples;

import java.util.Arrays;
import java.util.List;
import java.util.Map;

import org.apache.commons.lang3.tuple.Pair;
import org.apache.commons.math3.complex.Complex;

import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
import brainflow.DetrendOperations;
import brainflow.LogLevels;
import brainflow.WindowFunctions;

public class BandPower
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = BoardIds.SYNTHETIC_BOARD.get_code ();
        BoardShim board_shim = new BoardShim (board_id, params);
        Map<String, Object> board_descr = BoardShim.get_board_descr (board_id);
        int sampling_rate = ((Double) board_descr.get ("sampling_rate")).intValue ();
        int nfft = DataFilter.get_nearest_power_of_two (sampling_rate);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in␣
↪the main thread");
        Thread.sleep (10000);
        board_shim.stop_stream ();
        double[][] data = board_shim.get_board_data ();
```

(continues on next page)

(continued from previous page)

```
        board_shim.release_session ();

        @SuppressWarnings ("unchecked")
        List<Double> eeg_channels = (List<Double>) board_descr.get ("eeg_channels");
        // seconds channel of synthetic board has big 'alpha' use it for test
        int eeg_channel = eeg_channels.get (1).intValue ();
        // optional: detrend before psd
        DataFilter.detrend (data[eeg_channel], DetrendOperations.LINEAR.get_code ());
        Pair<double[], double[]> psd = DataFilter.get_psd_welch (data[eeg_channel],␣
↪nfft, nfft / 2, sampling_rate,
                WindowFunctions.HANNING.get_code ());
        double band_power_alpha = DataFilter.get_band_power (psd, 7.0, 13.0);
        double band_power_beta = DataFilter.get_band_power (psd, 14.0, 30.0);
        System.out.println ("Alpha/Beta Ratio: " + (band_power_alpha / band_power_
↪beta));
    }
}
```

5.2.9 Java EEG Metrics

```
package brainflow.examples;

import org.apache.commons.lang3.ArrayUtils;
import org.apache.commons.lang3.tuple.Pair;

import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowClassifiers;
import brainflow.BrainFlowInputParams;
import brainflow.BrainFlowMetrics;
import brainflow.BrainFlowModelParams;
import brainflow.DataFilter;
import brainflow.LogLevels;
import brainflow.MLModel;

public class EEGMetrics
{

    public static void main (String[] args) throws Exception
    {
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        int board_id = parse_args (args, params);
        BoardShim board_shim = new BoardShim (board_id, params);
        int master_board_id = board_shim.get_board_id ();
        int sampling_rate = BoardShim.get_sampling_rate (master_board_id);
        int[] eeg_channels = BoardShim.get_eeg_channels (master_board_id);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO.get_code (), "Start sleeping in␣
↪the main thread");
        // recommended window size for eeg metric calculation is at least 4 seconds,
        // bigger is better
        Thread.sleep (5000);
```

(continues on next page)

(continued from previous page)

```
        board_shim.stop_stream ();
        double[][] data = board_shim.get_board_data ();
        board_shim.release_session ();

        Pair<double[], double[]> bands = DataFilter.get_avg_band_powers (data, eeg_
↪channels, sampling_rate, true);
        double[] feature_vector = ArrayUtils.addAll (bands.getLeft (), bands.getRight␣
↪());
        BrainFlowModelParams model_params = new BrainFlowModelParams␣
↪(BrainFlowMetrics.CONCENTRATION.get_code (),
                BrainFlowClassifiers.REGRESSION.get_code ());
        MLModel concentration = new MLModel (model_params);
        concentration.prepare ();
        System.out.print ("Concentration: " + concentration.predict (feature_vector));
        concentration.release ();
    }

    private static int parse_args (String[] args, BrainFlowInputParams params)
    {
        int board_id = -1;
        for (int i = 0; i < args.length; i++)
        {
            if (args[i].equals ("--ip-address"))
            {
                params.ip_address = args[i + 1];
            }
            if (args[i].equals ("--serial-port"))
            {
                params.serial_port = args[i + 1];
            }
            if (args[i].equals ("--ip-port"))
            {
                params.ip_port = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--ip-protocol"))
            {
                params.ip_protocol = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--other-info"))
            {
                params.other_info = args[i + 1];
            }
            if (args[i].equals ("--board-id"))
            {
                board_id = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--timeout"))
            {
                params.timeout = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--serial-number"))
            {
                params.serial_number = args[i + 1];
            }
            if (args[i].equals ("--file"))
            {
                params.file = args[i + 1];
```

(continues on next page)

(continued from previous page)

```
            }
        }
        return board_id;
    }
}
```

5.3 C#

5.3.1 C# Read Data from a Board

```
using System;

using brainflow;
using brainflow.math;


namespace test
{
    class GetBoardData
    {
        static void Main (string[] args)
        {
            BoardShim.enable_dev_board_logger ();

            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = parse_args (args, input_params);

            BoardShim board_shim = new BoardShim (board_id, input_params);
            board_shim.prepare_session ();
            // board_shim.start_stream (); // use this for default options
            board_shim.start_stream (450000, "file://file_stream.csv:w");
            System.Threading.Thread.Sleep (5000);
            board_shim.stop_stream ();
            double[,] unprocessed_data = board_shim.get_current_board_data (20);
            int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
            foreach (var index in eeg_channels)
                Console.WriteLine ("[{0}]", string.Join (", ", unprocessed_data.
↪GetRow (index)));
            board_shim.release_session ();
        }

        static int parse_args (string[] args, BrainFlowInputParams input_params)
        {
            int board_id = (int)BoardIds.SYNTHETIC_BOARD; //assume synthetic board by␣
↪default
            // use docs to get params for your specific board, e.g. set serial_port␣
↪for Cyton
            for (int i = 0; i < args.Length; i++)
            {
                if (args[i].Equals ("--ip-address"))
                {
                    input_params.ip_address = args[i + 1];
                }
```

(continues on next page)

(continued from previous page)

```
                if (args[i].Equals ("--mac-address"))
                {
                    input_params.mac_address = args[i + 1];
                }
                if (args[i].Equals ("--serial-port"))
                {
                    input_params.serial_port = args[i + 1];
                }
                if (args[i].Equals ("--other-info"))
                {
                    input_params.other_info = args[i + 1];
                }
                if (args[i].Equals ("--ip-port"))
                {
                    input_params.ip_port = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--ip-protocol"))
                {
                    input_params.ip_protocol = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--board-id"))
                {
                    board_id = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals("--timeout"))
                {
                    input_params.timeout = Convert.ToInt32(args[i + 1]);
                }
                if (args[i].Equals("--serial-number"))
                {
                    input_params.serial_number = args[i + 1];
                }
                if (args[i].Equals ("--file"))
                {
                    input_params.file = args[i + 1];
                }
            }
            return board_id;
        }
    }
}
```

5.3.2 C# Markers

```
using System;

using brainflow;
using brainflow.math;


namespace test
{
    class Markers
    {
        static void Main (string[] args)
```

(continues on next page)

(continued from previous page)

```
        {
            BoardShim.enable_dev_board_logger ();

            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = parse_args (args, input_params);

            BoardShim board_shim = new BoardShim (board_id, input_params);
            board_shim.prepare_session ();
            board_shim.start_stream (450000, "file://file_stream.csv:w");
            for (int i = 1; i < 5; i++)
            {
                System.Threading.Thread.Sleep (1000);
                board_shim.insert_marker (i);
            }
            board_shim.stop_stream ();
            board_shim.release_session ();
        }

        static int parse_args (string[] args, BrainFlowInputParams input_params)
        {
            int board_id = (int)BoardIds.SYNTHETIC_BOARD; //assume synthetic board by␣
↪default
            // use docs to get params for your specific board, e.g. set serial_port␣
↪for Cyton
            for (int i = 0; i < args.Length; i++)
            {
                if (args[i].Equals ("--ip-address"))
                {
                    input_params.ip_address = args[i + 1];
                }
                if (args[i].Equals ("--mac-address"))
                {
                    input_params.mac_address = args[i + 1];
                }
                if (args[i].Equals ("--serial-port"))
                {
                    input_params.serial_port = args[i + 1];
                }
                if (args[i].Equals ("--other-info"))
                {
                    input_params.other_info = args[i + 1];
                }
                if (args[i].Equals ("--ip-port"))
                {
                    input_params.ip_port = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--ip-protocol"))
                {
                    input_params.ip_protocol = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--board-id"))
                {
                    board_id = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals("--timeout"))
                {
                    input_params.timeout = Convert.ToInt32(args[i + 1]);
```

(continues on next page)

(continued from previous page)

```
                }
                if (args[i].Equals("--serial-number"))
                {
                    input_params.serial_number = args[i + 1];
                }
                if (args[i].Equals ("--file"))
                {
                    input_params.file = args[i + 1];
                }
            }
            return board_id;
        }
    }
}
```

5.3.3 C# Read Write File

```
using System;

using brainflow;
using brainflow.math;


namespace test
{
    class Serialization
    {
        static void Main (string[] args)
        {
            // use synthetic board for demo
            BoardShim.enable_dev_board_logger ();
            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = (int)BoardIds.SYNTHETIC_BOARD;

            BoardShim board_shim = new BoardShim (board_id, input_params);
            board_shim.prepare_session ();
            board_shim.start_stream (3600);
            System.Threading.Thread.Sleep (5000);
            board_shim.stop_stream ();
            double[,] unprocessed_data = board_shim.get_current_board_data (20);
            int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
            Console.WriteLine ("Before serialization:");
            foreach (var index in eeg_channels)
                Console.WriteLine ("[{0}]", string.Join (", ", unprocessed_data.
↪GetRow (index)));
            board_shim.release_session ();

            // demo for data serialization
            DataFilter.write_file (unprocessed_data, "test.csv", "w");
            double[,] restored_data = DataFilter.read_file ("test.csv");
            Console.WriteLine ("After Serialization:");
            foreach (var index in eeg_channels)
                Console.WriteLine ("[{0}]", string.Join (", ", restored_data.GetRow␣
↪(index)));
        }
```

(continues on next page)

(continued from previous page)

```
    }
}
```

5.3.4 C# Downsample Data

```
using System;

using brainflow;
using brainflow.math;


namespace test
{
    class Downsampling
    {
        static void Main (string[] args)
        {
            // use synthetic board for demo
            BoardShim.enable_dev_board_logger ();
            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = (int)BoardIds.SYNTHETIC_BOARD;

            BoardShim board_shim = new BoardShim (board_id, input_params);
            board_shim.prepare_session ();
            board_shim.start_stream (3600);
            System.Threading.Thread.Sleep (5000);
            board_shim.stop_stream ();
            double[,] unprocessed_data = board_shim.get_current_board_data (20);
            int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
            board_shim.release_session ();

            for (int i = 0; i < eeg_channels.Length; i++)
            {
                Console.WriteLine ("Before processing:");
                Console.WriteLine ("[{0}]", string.Join (", ", unprocessed_data.
↪GetRow(eeg_channels[i])));
                // you can use MEAN, MEDIAN or EACH for downsampling
                double[] filtered = DataFilter.perform_downsampling (unprocessed_data.
↪GetRow (eeg_channels[i]), 3, (int)AggOperations.MEDIAN);
                Console.WriteLine ("Before processing:");
                Console.WriteLine ("[{0}]", string.Join (", ", filtered));
            }
        }
    }
}
```

5.3.5 C# Transforms

```
using System;
using System.Numerics;

using brainflow;
using brainflow.math;


namespace test
{
    class Transforms
    {
        static void Main (string[] args)
        {
            // use synthetic board for demo
            BoardShim.enable_dev_board_logger ();
            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = (int)BoardIds.SYNTHETIC_BOARD;

            BoardShim board_shim = new BoardShim (board_id, input_params);
            board_shim.prepare_session ();
            board_shim.start_stream (3600);
            System.Threading.Thread.Sleep (5000);
            board_shim.stop_stream ();
            double[,] unprocessed_data = board_shim.get_current_board_data (64);
            int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
            board_shim.release_session ();

            for (int i = 0; i < eeg_channels.Length; i++)
            {
                Console.WriteLine ("Original data:");
                Console.WriteLine ("[{0}]", string.Join (", ", unprocessed_data.
↪GetRow (eeg_channels[i])));
                // demo for wavelet transform
                // tuple of coeffs array in format[A(J) D(J) D(J-1) ..... D(1)] where␣
↪J is a
                // decomposition level, A - app coeffs, D - detailed coeffs, and␣
↪array which stores
                // length for each block, len of this array is decomposition_length +␣
↪1
                Tuple<double[], int[]> wavelet_data = DataFilter.perform_wavelet_
↪transform(unprocessed_data.GetRow (eeg_channels[i]), "db4", 3);
                // print app coeffs
                for (int j = 0; j < wavelet_data.Item2[0]; j++)
                {
                    Console.Write (wavelet_data.Item1[j] + " ");
                }
                Console.WriteLine ();
                // you can do smth with wavelet coeffs here, for example denoising␣
↪works via thresholds for wavelets coeffs
                double[] restored_data = DataFilter.perform_inverse_wavelet_transform␣
↪(wavelet_data, unprocessed_data.GetRow (eeg_channels[i]).Length, "db4", 3);
                Console.WriteLine ("Restored wavelet data:");
                Console.WriteLine ("[{0}]", string.Join (", ", restored_data));

                // demo for fft
```

(continues on next page)

(continued from previous page)

```
                // end_pos - start_pos must be a power of 2
                Complex[] fft_data = DataFilter.perform_fft (unprocessed_data.GetRow␣
↪(eeg_channels[i]), 0, 64, (int)WindowFunctions.HAMMING);
                // len of fft_data is N / 2 + 1
                double[] restored_fft_data = DataFilter.perform_ifft (fft_data);
                Console.WriteLine ("Restored fft data:");
                Console.WriteLine ("[{0}]", string.Join (", ", restored_fft_data));
            }
        }
    }
}
```

5.3.6 C# Signal Filtering

```
using System;

using brainflow;
using brainflow.math;


namespace test
{
    class SignalFiltering
    {
        static void Main (string[] args)
        {
            // use synthetic board for demo
            BoardShim.enable_dev_board_logger ();
            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = (int)BoardIds.SYNTHETIC_BOARD;

            BoardShim board_shim = new BoardShim (board_id, input_params);
            board_shim.prepare_session ();
            board_shim.start_stream (3600);
            System.Threading.Thread.Sleep (5000);
            board_shim.stop_stream ();
            double[,] unprocessed_data = board_shim.get_current_board_data (20);
            int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
            board_shim.release_session ();

            // for demo apply different filters to different channels
            double[] filtered;
            for (int i = 0; i < eeg_channels.Length; i++)
            {
                Console.WriteLine ("Before processing:");
                Console.WriteLine ("[{0}]", string.Join (", ", unprocessed_data.
↪GetRow (eeg_channels[i])));
                switch (i)
                {
                    case 0:
                        filtered = DataFilter.perform_lowpass (unprocessed_data.
↪GetRow(eeg_channels[i]), BoardShim.get_sampling_rate (board_id), 20.0, 4,␣
↪(int)FilterTypes.BESSEL, 0.0);
                        Console.WriteLine ("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine ("[{0}]", string.Join (", ", filtered));
```

(continues on next page)

(continued from previous page)

```
                        break;
                    case 1:
                        filtered = DataFilter.perform_highpass (unprocessed_data.
↪GetRow (eeg_channels[i]), BoardShim.get_sampling_rate (board_id), 2.0, 4,␣
↪(int)FilterTypes.BUTTERWORTH, 0.0);
                        Console.WriteLine ("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine ("[{0}]", string.Join (", ", filtered));
                        break;
                    case 2:
                        filtered = DataFilter.perform_bandpass (unprocessed_data.
↪GetRow (eeg_channels[i]), BoardShim.get_sampling_rate (board_id), 15.0, 5.0, 2,␣
↪(int)FilterTypes.BUTTERWORTH, 0.0);
                        Console.WriteLine ("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine ("[{0}]", string.Join (", ", filtered));
                        break;
                    case 3:
                        filtered = DataFilter.perform_bandstop (unprocessed_data.
↪GetRow (eeg_channels[i]), BoardShim.get_sampling_rate (board_id), 50.0, 1.0, 6,␣
↪(int)FilterTypes.CHEBYSHEV_TYPE_1, 1.0);
                        Console.WriteLine ("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine ("[{0}]", string.Join (", ", filtered));
                        break;
                    default:
                        filtered = DataFilter.remove_environmental_noise(unprocessed_
↪data.GetRow(eeg_channels[i]), BoardShim.get_sampling_rate(board_id),␣
↪(int)NoiseTypes.FIFTY);
                        Console.WriteLine("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine("[{0}]", string.Join(", ", filtered));
                        break;
                }
            }
        }
    }
}
```

5.3.7 C# Denoising

```
using System;

using brainflow;
using brainflow.math;


namespace test
{
    class Denoising
    {
        static void Main (string[] args)
        {
            // use synthetic board for demo
            BoardShim.enable_dev_board_logger ();
            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = (int)BoardIds.SYNTHETIC_BOARD;

            BoardShim board_shim = new BoardShim (board_id, input_params);
```

(continues on next page)

(continued from previous page)

```
            board_shim.prepare_session ();
            board_shim.start_stream (3600);
            System.Threading.Thread.Sleep (5000);
            board_shim.stop_stream ();
            double[,] unprocessed_data = board_shim.get_current_board_data (64);
            int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
            foreach (var index in eeg_channels)
                Console.WriteLine ("[{0}]", string.Join (", ", unprocessed_data.
↪GetRow (index)));
            board_shim.release_session ();

            // for demo apply different methods to different channels
            double[] filtered;
            for (int i = 0; i < eeg_channels.Length; i++)
            {
                switch (i)
                {
                    // first of all you can try simple moving average or moving median
                    case 0:
                        filtered = DataFilter.perform_rolling_filter (unprocessed_
↪data.GetRow (eeg_channels[i]), 3, (int)AggOperations.MEAN);
                        Console.WriteLine ("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine ("[{0}]", string.Join (", ", filtered));
                        break;
                    case 1:
                        filtered = DataFilter.perform_rolling_filter (unprocessed_
↪data.GetRow (eeg_channels[i]), 3, (int)AggOperations.MEDIAN);
                        Console.WriteLine ("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine ("[{0}]", string.Join (", ", filtered));
                        break;
                    // if for your signal these methods dont work good you can try␣
↪wavelet based denoising
                    default:
                        // feel free to try different functions and different␣
↪decomposition levels
                        filtered = DataFilter.perform_wavelet_denoising (unprocessed_
↪data.GetRow (eeg_channels[i]), "db4", 3);
                        Console.WriteLine ("Filtered channel " + eeg_channels[i]);
                        Console.WriteLine ("[{0}]", string.Join (", ", filtered));
                        break;
                }
            }
        }
    }
}
```

5.3.8 C# Band Power

```
using System;
using System.Runtime.Serialization;
using brainflow;
using brainflow.math;


namespace test
{
    class BandPower
    {
        static void Main (string[] args)
        {
            // use synthetic board for demo
            BoardShim.enable_dev_board_logger ();
            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = (int)BoardIds.SYNTHETIC_BOARD;
            BoardDescr board_descr = BoardShim.get_board_descr<BoardDescr>(board_id);
            int sampling_rate = board_descr.sampling_rate;
            int nfft = DataFilter.get_nearest_power_of_two(sampling_rate);

            BoardShim board_shim = new BoardShim (board_id, input_params);
            board_shim.prepare_session ();
            board_shim.start_stream (3600);
            System.Threading.Thread.Sleep (10000);
            board_shim.stop_stream ();
            double[,] data = board_shim.get_board_data ();
            int[] eeg_channels = board_descr.eeg_channels;
            // use second channel of synthetic board to see 'alpha'
            int channel = eeg_channels[1];
            board_shim.release_session ();
            double[] detrend = DataFilter.detrend(data.GetRow(channel),␣
↪(int)DetrendOperations.LINEAR);
            Tuple<double[], double[]> psd = DataFilter.get_psd_welch (detrend, nfft,␣
↪nfft / 2, sampling_rate, (int)WindowFunctions.HANNING);
            double band_power_alpha = DataFilter.get_band_power (psd, 7.0, 13.0);
            double band_power_beta = DataFilter.get_band_power (psd, 14.0, 30.0);
            Console.WriteLine ("Alpha/Beta Ratio:" + (band_power_alpha/ band_power_
↪beta));
        }
    }
}
```

5.3.9 C# EEG Metrics

```
using System;

using brainflow;
using brainflow.math;


namespace test
{
    class EEGMetrics
```

(continues on next page)

(continued from previous page)

```
    {
        static void Main (string[] args)
        {
            // use synthetic board for demo
            BoardShim.enable_dev_board_logger ();
            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            int board_id = parse_args (args, input_params);
            BoardShim board_shim = new BoardShim (board_id, input_params);
            int sampling_rate = BoardShim.get_sampling_rate (board_shim.get_board_id␣
↪());
            int nfft = DataFilter.get_nearest_power_of_two (sampling_rate);
            int[] eeg_channels = BoardShim.get_eeg_channels (board_shim.get_board_id␣
↪());

            board_shim.prepare_session ();
            board_shim.start_stream (3600);
            System.Threading.Thread.Sleep (10000);
            board_shim.stop_stream ();
            double[,] data = board_shim.get_board_data ();
            board_shim.release_session ();

            Tuple<double[], double[]> bands = DataFilter.get_avg_band_powers (data,␣
↪eeg_channels, sampling_rate, true);
            double[] feature_vector = bands.Item1.Concatenate (bands.Item2);
            BrainFlowModelParams model_params = new BrainFlowModelParams␣
↪((int)BrainFlowMetrics.CONCENTRATION, (int)BrainFlowClassifiers.REGRESSION);
            MLModel concentration = new MLModel (model_params);
            concentration.prepare ();
            Console.WriteLine ("Concentration: " + concentration.predict (feature_
↪vector));
            concentration.release ();
        }

        static int parse_args (string[] args, BrainFlowInputParams input_params)
        {
            int board_id = (int)BoardIds.SYNTHETIC_BOARD; //assume synthetic board by␣
↪default
            // use docs to get params for your specific board, e.g. set serial_port␣
↪for Cyton
            for (int i = 0; i < args.Length; i++)
            {
                if (args[i].Equals ("--ip-address"))
                {
                    input_params.ip_address = args[i + 1];
                }
                if (args[i].Equals ("--mac-address"))
                {
                    input_params.mac_address = args[i + 1];
                }
                if (args[i].Equals ("--serial-port"))
                {
                    input_params.serial_port = args[i + 1];
                }
                if (args[i].Equals ("--other-info"))
                {
                    input_params.other_info = args[i + 1];
                }
```

(continues on next page)

(continued from previous page)

```
                if (args[i].Equals ("--ip-port"))
                {
                    input_params.ip_port = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--ip-protocol"))
                {
                    input_params.ip_protocol = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--board-id"))
                {
                    board_id = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--timeout"))
                {
                    input_params.timeout = Convert.ToInt32 (args[i + 1]);
                }
                if (args[i].Equals ("--serial-number"))
                {
                    input_params.serial_number = args[i + 1];
                }
                if (args[i].Equals ("--file"))
                {
                    input_params.file = args[i + 1];
                }
            }
            return board_id;
        }
    }
}
```

5.4 C++

To compile examples below for Linux or MacOS run:

```
cd tests/cpp/get_data_demo
mkdir build
cd build
cmake -DCMAKE_PREFIX_PATH=TYPE_FULL_PATH_TO_BRAINFLOW_INSTALLED_FOLDER ..
# e.g. cmake -DCMAKE_PREFIX_PATH=/home/andrey/brainflow/installed_linux ..
make
```

For Windows it's almost the same.

Make sure that compiled dynamic libraries exist in search path before running an executable by doing one of the following:

- for Linux and MacOS add them to LD_LIBRARY_PATH env variable
- for Windows add them to PATH env variable
- or just copypaste them to the folder where your executable is located

5.4.1 CMake File Example

```
cmake_minimum_required (VERSION 3.10)
project (BRAINFLOW_GET_DATA)

set (CMAKE_CXX_STANDARD 11)
set (CMAKE_VERBOSE_MAKEFILE ON)

macro (configure_msvc_runtime)
    if (MSVC)
        # Default to statically-linked runtime.
        if ("${MSVC_RUNTIME}" STREQUAL "")
            set (MSVC_RUNTIME "static")
        endif ()
        # Set compiler options.
        set (variables
            CMAKE_C_FLAGS_DEBUG
            CMAKE_C_FLAGS_MINSIZEREL
            CMAKE_C_FLAGS_RELEASE
            CMAKE_C_FLAGS_RELWITHDEBINFO
            CMAKE_CXX_FLAGS_DEBUG
            CMAKE_CXX_FLAGS_MINSIZEREL
            CMAKE_CXX_FLAGS_RELEASE
            CMAKE_CXX_FLAGS_RELWITHDEBINFO
        )
        if (${MSVC_RUNTIME} STREQUAL "static")
            message(STATUS
                "MSVC -> forcing use of statically-linked runtime."
            )
            foreach (variable ${variables})
                if (${variable} MATCHES "/MD")
                    string (REGEX REPLACE "/MD" "/MT" ${variable} "${${variable}}")
                endif ()
            endforeach ()
        else ()
            message (STATUS
                "MSVC -> forcing use of dynamically-linked runtime."
            )
            foreach (variable ${variables})
                if (${variable} MATCHES "/MT")
                    string (REGEX REPLACE "/MT" "/MD" ${variable} "${${variable}}")
                endif ()
            endforeach ()
        endif ()
    endif ()
endmacro ()

# link msvc runtime statically
configure_msvc_runtime()

find_package (
    brainflow CONFIG REQUIRED
)

add_executable (
    brainflow_get_data
    src/brainflow_get_data.cpp
```

(continues on next page)

(continued from previous page)

```
)

target_include_directories (
    brainflow_get_data PUBLIC
    ${brainflow_INCLUDE_DIRS}
)

target_link_libraries (
    brainflow_get_data PUBLIC
    # for some systems(ubuntu for example) order matters
    ${BrainflowPath}
    ${MLModulePath}
    ${DataHandlerPath}
    ${BoardControllerPath}
)

add_executable (
    markers
    src/markers.cpp
)

target_include_directories (
    markers PUBLIC
    ${brainflow_INCLUDE_DIRS}
)

target_link_libraries (
    markers PUBLIC
    # for some systems(ubuntu for example) order matters
    ${BrainflowPath}
    ${MLModulePath}
    ${DataHandlerPath}
    ${BoardControllerPath}
)
```

5.4.2 C++ Read Data from a Board

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"

using namespace std;

bool parse_args (int argc, char *argv[], struct BrainFlowInputParams *params, int␣
↪*board_id);
```

(continues on next page)

(continued from previous page)

```
int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int board_id = 0;
    if (!parse_args (argc, argv, &params, &board_id))
    {
        return -1;
    }
    int res = 0;

    BoardShim *board = new BoardShim (board_id, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
        Sleep (5000);
#else
        sleep (5);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_current_board_data (10);
        board->release_session ();
        std::cout << data << std::endl;
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    return res;
}

bool parse_args (int argc, char *argv[], struct BrainFlowInputParams *params, int␣
↪*board_id)
{
    bool board_id_found = false;
    for (int i = 1; i < argc; i++)
    {
        if (std::string (argv[i]) == std::string ("--board-id"))
        {
            if (i + 1 < argc)
            {
                i++;
                board_id_found = true;
```

(continues on next page)

(continued from previous page)

```
                *board_id = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-address"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_address = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-port"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_port = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--serial-port"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->serial_port = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-protocol"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_protocol = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
```

(continues on next page)

(continued from previous page)

```
            return false;
        }
    }
    if (std::string (argv[i]) == std::string ("--timeout"))
    {
        if (i + 1 < argc)
        {
            i++;
            params->timeout = std::stoi (std::string (argv[i]));
        }
        else
        {
            std::cerr << "missed argument" << std::endl;
            return false;
        }
    }
    if (std::string (argv[i]) == std::string ("--other-info"))
    {
        if (i + 1 < argc)
        {
            i++;
            params->other_info = std::string (argv[i]);
        }
        else
        {
            std::cerr << "missed argument" << std::endl;
            return false;
        }
    }
    if (std::string (argv[i]) == std::string ("--mac-address"))
    {
        if (i + 1 < argc)
        {
            i++;
            params->mac_address = std::string (argv[i]);
        }
        else
        {
            std::cerr << "missed argument" << std::endl;
            return false;
        }
    }
    if (std::string (argv[i]) == std::string ("--serial-number"))
    {
        if (i + 1 < argc)
        {
            i++;
            params->serial_number = std::string (argv[i]);
        }
        else
        {
            std::cerr << "missed argument" << std::endl;
            return false;
        }
    }
    if (std::string (argv[i]) == std::string ("--file"))
    {
```

(continues on next page)

(continued from previous page)

```
            if (i + 1 < argc)
            {
                i++;
                params->file = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
    }
    if (!board_id_found)
    {
        std::cerr << "board id is not provided" << std::endl;
        return false;
    }
    return true;
}
```

5.4.3 C++ Markers

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"

using namespace std;

bool parse_args (int argc, char *argv[], struct BrainFlowInputParams *params, int␣
↪*board_id);


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int board_id = 0;
    if (!parse_args (argc, argv, &params, &board_id))
    {
        return -1;
    }
    int res = 0;

    BoardShim *board = new BoardShim (board_id, params);

    try
```

(continues on next page)

(continued from previous page)

```
    {
        board->prepare_session ();
        board->start_stream ();

        for (int i = 1; i < 5; i++)
        {
            board->insert_marker (i);
#ifdef _WIN32
            Sleep (1000);
#else
            sleep (1);
#endif
        }

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_board_data ();
        board->release_session ();
        std::cout << data << std::endl;
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    return res;
}

bool parse_args (int argc, char *argv[], struct BrainFlowInputParams *params, int␣
↪*board_id)
{
    bool board_id_found = false;
    for (int i = 1; i < argc; i++)
    {
        if (std::string (argv[i]) == std::string ("--board-id"))
        {
            if (i + 1 < argc)
            {
                i++;
                board_id_found = true;
                *board_id = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-address"))
        {
            if (i + 1 < argc)
```

(continues on next page)

(continued from previous page)

```
            {
                i++;
                params->ip_address = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-port"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_port = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--serial-port"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->serial_port = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-protocol"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_protocol = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--timeout"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->timeout = std::stoi (std::string (argv[i]));
            }
            else
```

(continues on next page)

(continued from previous page)

```
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--other-info"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->other_info = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--mac-address"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->mac_address = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--serial-number"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->serial_number = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--file"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->file = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
```

(continues on next page)

(continued from previous page)

```
    }
    if (!board_id_found)
    {
        std::cerr << "board id is not provided" << std::endl;
        return false;
    }
    return true;
}
```

5.4.4 C++ Read Write File

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"
#include "data_filter.h"

using namespace std;


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int res = 0;
    int board_id = (int)BoardIds::SYNTHETIC_BOARD;
    // use synthetic board for demo
    BoardShim *board = new BoardShim (board_id, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
        Sleep (5000);
#else
        sleep (5);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_current_board_data (10);
        board->release_session ();
        std::cout << "Original data:" << std::endl << data << std::endl;
        DataFilter::write_file (data, "test.csv", "w");
        BrainFlowArray<double, 2> restored_data = DataFilter::read_file ("test.csv");
        std::cout << "Restored data:" << std::endl << restored_data << std::endl;
```

(continues on next page)

(continued from previous page)

```
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    return res;
}
```

5.4.5 C++ Downsample Data

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"
#include "data_filter.h"

using namespace std;

void print_one_row (double *data, int num_data_points);


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int res = 0;
    int board_id = (int)BoardIds::SYNTHETIC_BOARD;
    // use synthetic board for demo
    BoardShim *board = new BoardShim (board_id, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
        Sleep (5000);
#else
        sleep (5);
```

(continues on next page)

(continued from previous page)

```
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_current_board_data (32);
        board->release_session ();

        double *downsampled_data = NULL;
        int filtered_size = 0;
        std::vector<int> eeg_channels = BoardShim::get_eeg_channels (board_id);

        for (int i = 0; i < eeg_channels.size (); i++)
        {
            std::cout << "Data from :" << eeg_channels[i] << " before downsampling " <
↪< std::endl;
            print_one_row (data.get_address (eeg_channels[i]), data.get_size (1));

            // just for demo apply different downsampling algorithms to different␣
↪channels
            // downsampling here just aggregates data points
            switch (i)
            {
                case 0:
                    downsampled_data =
                        DataFilter::perform_downsampling (data.get_address (eeg_
↪channels[i]),
                            data.get_size (1), 2, (int)AggOperations::MEAN, &filtered_
↪size);
                    break;
                case 1:
                    downsampled_data =
                        DataFilter::perform_downsampling (data.get_address (eeg_
↪channels[i]),
                            data.get_size (1), 3, (int)AggOperations::MEDIAN, &
↪filtered_size);
                    break;
                default:
                    downsampled_data =
                        DataFilter::perform_downsampling (data.get_address (eeg_
↪channels[i]),
                            data.get_size (1), 2, (int)AggOperations::EACH, &filtered_
↪size);
                    break;
            }

            std::cout << "Data from :" << eeg_channels[i] << " after downsampling " <
↪< std::endl;
            print_one_row (downsampled_data, filtered_size);
            delete[] downsampled_data;
        }
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
```

(continues on next page)

(continued from previous page)

```
        }
    }

    delete board;

    return res;
}

void print_one_row (double *data, int num_data_points)
{
    // print only first 10 data points
    int num_points = (num_data_points < 10) ? num_data_points : 10;
    for (int i = 0; i < num_points; i++)
    {
        std::cout << data[i] << " ";
    }
    std::cout << std::endl;
}
```

5.4.6 C++ Transforms

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"
#include "data_filter.h"

using namespace std;


void print_one_row (double *data, int num_data_points);


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int res = 0;
    // use synthetic board for demo
    BoardShim *board = new BoardShim ((int)BoardIds::SYNTHETIC_BOARD, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
```

(continues on next page)

(continued from previous page)

```
        Sleep (10000);
#else
        sleep (10);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_current_board_data (128);
        board->release_session ();
        std::cout << "Original data:" << std::endl << data << std::endl;

        // apply filters
        int sampling_rate = BoardShim::get_sampling_rate ((int)BoardIds::SYNTHETIC_
↪BOARD);
        std::vector<int> eeg_channels =
            BoardShim::get_eeg_channels ((int)BoardIds::SYNTHETIC_BOARD);
        int data_count = data.get_size (1);
        for (int i = 0; i < eeg_channels.size (); i++)
        {
            // demo for wavelet transform
            // std::pair of coeffs array in format[A(J) D(J) D(J-1) ..... D(1)] where␣
↪J is a
            // decomposition level, A - app coeffs, D - detailed coeffs, and array␣
↪which stores
            // length for each block, len of this array is decomposition_length + 1
            std::pair<double *, int *> wavelet_output = DataFilter::perform_wavelet_
↪transform (
                data.get_address (eeg_channels[i]), data_count, "db4", 4);
            // you can do smth with wavelet coeffs here, for example denoising works␣
↪via thresholds
            // for wavelet coefficients
            std::cout << "approximation coefficients:" << std::endl;
            for (int i = 0; i < wavelet_output.second[0]; i++)
            {
                std::cout << wavelet_output.first[i] << " ";
            }
            std::cout << std::endl;
            std::cout << "first block of detailed coefficients:" << std::endl;
            for (int i = wavelet_output.second[0];
                 i < wavelet_output.second[0] + wavelet_output.second[1]; i++)
            {
                std::cout << wavelet_output.first[i] << " ";
            }
            std::cout << std::endl;

            double *restored_data = DataFilter::perform_inverse_wavelet_transform (
                wavelet_output, data_count, "db4", 4);

            std::cout << "Original data:" << std::endl;
            print_one_row (data.get_address (eeg_channels[i]), data_count);
            std::cout << "Restored after inverse wavelet transform data:" << std::
↪endl;
            print_one_row (restored_data, data_count);

            delete[] wavelet_output.first;
            delete[] restored_data;
            delete[] wavelet_output.second;
```

(continues on next page)

(continued from previous page)

```
            // demo for fft
            // data count must be power of 2 for fft!
            std::complex<double> *fft_data = DataFilter::perform_fft (
                data.get_address (eeg_channels[i]), data_count, (int)WindowFunctions::
↪NO_WINDOW);
            // len of fft_data array is N / 2 + 1
            std::cout << "FFT coeffs:" << std::endl;
            for (int i = 0; i < data_count / 2 + 1; i++)
            {
                std::cout << fft_data[i] << " ";
            }
            std::cout << std::endl;
            double *restored_from_fft_data = DataFilter::perform_ifft (fft_data, data_
↪count);
            std::cout << "Restored after inverse fft transform data:" << std::endl;
            print_one_row (restored_from_fft_data, data_count);

            delete[] fft_data;
            delete[] restored_from_fft_data;
        }
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    return res;
}

void print_one_row (double *data, int num_data_points)
{
    for (int i = 0; i < num_data_points; i++)
    {
        std::cout << data[i] << " ";
    }
    std::cout << std::endl;
}
```

5.4.7 C++ Signal Filtering

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
```

(continues on next page)

(continued from previous page)

```
#endif

#include "board_shim.h"
#include "data_filter.h"

using namespace std;


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int res = 0;
    int board_id = (int)BoardIds::SYNTHETIC_BOARD;
    // use synthetic board for demo
    BoardShim *board = new BoardShim (board_id, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
        Sleep (5000);
#else
        sleep (5);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_board_data ();
        board->release_session ();
        std::cout << "Original data:" << std::endl << data << std::endl;

        // apply filters
        int sampling_rate = BoardShim::get_sampling_rate ((int)BoardIds::SYNTHETIC_
↪BOARD);
        std::vector<int> eeg_channels = BoardShim::get_eeg_channels (board_id);
        for (int i = 0; i < eeg_channels.size (); i++)
        {
            switch (i)
            {
                // just for test and demo - apply different filters to different eeg␣
↪channels
                // signal filtering methods work in-place
                case 0:
                    DataFilter::perform_lowpass (data.get_address (eeg_channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id),␣
↪30.0, 3,
                        (int)FilterTypes::BUTTERWORTH, 0);
                    break;
                case 1:
                    DataFilter::perform_highpass (data.get_address (eeg_channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id), 5.
↪0, 5,
                        (int)FilterTypes::CHEBYSHEV_TYPE_1, 1);
                    break;
```

(continues on next page)

(continued from previous page)

```
                case 2:
                    DataFilter::perform_bandpass (data.get_address (eeg_channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id),␣
↪15.0, 10.0, 3,
                        (int)FilterTypes::BESSEL, 0);
                    break;
                case 3:
                    DataFilter::perform_bandstop (data.get_address (eeg_channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id),␣
↪50.0, 4.0, 4,
                        (int)FilterTypes::BUTTERWORTH, 0);
                    break;
                default:
                    DataFilter::remove_environmental_noise (data.get_address (eeg_
↪channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id),
                        (int)NoiseTypes::FIFTY);
                    break;
            }
        }
        std::cout << "Filtered data:" << std::endl << data << std::endl;
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    return res;
}
```

5.4.8 C++ Denoising

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"
#include "data_filter.h"

using namespace std;
```

(continues on next page)

(continued from previous page)

```
int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int res = 0;
    int board_id = (int)BoardIds::SYNTHETIC_BOARD;
    // use synthetic board for demo
    BoardShim *board = new BoardShim (board_id, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
        Sleep (5000);
#else
        sleep (5);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_board_data ();
        board->release_session ();
        std::cout << "Original data:" << std::endl << data << std::endl;

        // apply filters
        std::vector<int> eeg_channels = BoardShim::get_eeg_channels (board_id);
        for (int i = 0; i < eeg_channels.size (); i++)
        {
            switch (i)
            {
                // for demo apply different methods to different channels
                case 0:
                    DataFilter::perform_rolling_filter (data.get_address (eeg_
↪channels[i]),
                        data.get_size (1), 3, (int)AggOperations::MEDIAN);
                    break;
                case 1:
                    DataFilter::perform_rolling_filter (data.get_address (eeg_
↪channels[i]),
                        data.get_size (1), 3, (int)AggOperations::MEAN);
                    break;
                case 2:
                    DataFilter::perform_rolling_filter (data.get_address (eeg_
↪channels[i]),
                        data.get_size (1), 5, (int)AggOperations::MEDIAN);
                    break;
                case 3:
                    DataFilter::perform_rolling_filter (data.get_address (eeg_
↪channels[i]),
                        data.get_size (1), 5, (int)AggOperations::MEAN);
                    break;
                    // if moving average and moving median dont work well for your␣
↪signal you can
                    // try wavelet based denoising, feel free to try different␣
↪wavelet functions and
```

(continues on next page)

(continued from previous page)

```
                    // decomposition levels
                case 4:
                    DataFilter::perform_wavelet_denoising (
                        data.get_address (eeg_channels[i]), data.get_size (1), "db4",␣
↪3);
                    break;
                case 5:
                    DataFilter::perform_wavelet_denoising (
                        data.get_address (eeg_channels[i]), data.get_size (1), "coif3
↪", 3);
                    break;
            }
        }
        std::cout << "Data after denoising:" << std::endl << data << std::endl;
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    return res;
}
```

5.4.9 C++ Band Power

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"
#include "data_filter.h"

using namespace std;


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int res = 0;
    int board_id = (int)BoardIds::SYNTHETIC_BOARD;
```

(continues on next page)

(continued from previous page)

```
    // use synthetic board for demo
    BoardShim *board = new BoardShim (board_id, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
        Sleep (10000);
#else
        sleep (10);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_board_data ();
        board->release_session ();
        std::cout << "Original data:" << std::endl << data << std::endl;

        // calc band powers
        json board_descr = BoardShim::get_board_descr (board_id);
        int sampling_rate = (int)board_descr["sampling_rate"];
        int fft_len = DataFilter::get_nearest_power_of_two (sampling_rate);
        std::vector<int> eeg_channels = board_descr["eeg_channels"];
        // for synthetic board second channel is a sine wave at 10 Hz, should see big␣
↪alpha
        int channel = eeg_channels[1];
        // optional - detrend
        DataFilter::detrend (
            data.get_address (channel), data.get_size (1), (int)DetrendOperations::
↪LINEAR);
        std::cout << "Data after detrend:" << std::endl << data << std::endl;
        std::pair<double *, double *> psd = DataFilter::get_psd_welch (data.get_
↪address (channel),
            data.get_size (1), fft_len, fft_len / 2, sampling_rate,␣
↪(int)WindowFunctions::HANNING);
        // calc band power
        double band_power_alpha = DataFilter::get_band_power (psd, fft_len / 2 + 1, 7.
↪0, 13.0);
        double band_power_beta = DataFilter::get_band_power (psd, fft_len / 2 + 1, 14.
↪0, 30.0);
        std::cout << "alpha/beta:" << band_power_alpha / band_power_beta << std::endl;
        delete[] psd.first;
        delete[] psd.second;
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;
```

(continues on next page)

(continued from previous page)

```
    return res;
}
```

5.4.10 C++ EEG Metrics

```
#include <iostream>
#include <stdlib.h>
#include <string>

#ifdef _WIN32
#include <windows.h>
#else
#include <unistd.h>
#endif

#include "board_shim.h"
#include "data_filter.h"
#include "ml_model.h"

using namespace std;

bool parse_args (int argc, char *argv[], struct BrainFlowInputParams *params, int␣
↪*board_id);


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    struct BrainFlowInputParams params;
    int board_id = 0;
    if (!parse_args (argc, argv, &params, &board_id))
    {
        return -1;
    }
    int res = 0;

    BoardShim *board = new BoardShim (board_id, params);

    try
    {
        board->prepare_session ();
        board->start_stream ();

#ifdef _WIN32
        Sleep (5000);
#else
        sleep (5);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_board_data ();
        board->release_session ();
        std::cout << data << std::endl;
        // calc band powers
```

(continues on next page)

(continued from previous page)

```
        int sampling_rate = BoardShim::get_sampling_rate ((int)BoardIds::SYNTHETIC_
↪BOARD);
        std::vector<int> eeg_channels = BoardShim::get_eeg_channels (board_id);
        std::pair<double *, double *> bands =
            DataFilter::get_avg_band_powers (data, eeg_channels, sampling_rate, true);

        double feature_vector[10];
        for (int i = 0; i < 5; i++)
        {
            feature_vector[i] = bands.first[i];
            feature_vector[i + 5] = bands.second[i];
        }
        for (int i = 0; i < 10; i++)
        {
            std::cout << feature_vector[i] << " ";
        }
        std::cout << std::endl;

        struct BrainFlowModelParams conc_model_params (
            (int)BrainFlowMetrics::CONCENTRATION, (int)BrainFlowClassifiers::
↪REGRESSION);
        MLModel concentration_model (conc_model_params);
        concentration_model.prepare ();
        std::cout << "Concentration Regression :"
                  << concentration_model.predict (feature_vector, 10) << std::endl;
        concentration_model.release ();

        struct BrainFlowModelParams relax_model_params (
            (int)BrainFlowMetrics::RELAXATION, (int)BrainFlowClassifiers::KNN);
        MLModel relaxation_model (relax_model_params);
        relaxation_model.prepare ();
        std::cout << "Relaxation KNN :" << relaxation_model.predict (feature_vector,␣
↪10)
                  << std::endl;
        relaxation_model.release ();

        delete[] bands.first;
        delete[] bands.second;
    }
    catch (const BrainFlowException &err)
    {
        BoardShim::log_message ((int)LogLevels::LEVEL_ERROR, err.what ());
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    return res;
}

bool parse_args (int argc, char *argv[], struct BrainFlowInputParams *params, int␣
↪*board_id)
{
```

(continues on next page)

(continued from previous page)

```
    bool board_id_found = false;
    for (int i = 1; i < argc; i++)
    {
        if (std::string (argv[i]) == std::string ("--board-id"))
        {
            if (i + 1 < argc)
            {
                i++;
                board_id_found = true;
                *board_id = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-address"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_address = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-port"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_port = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--serial-port"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->serial_port = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-protocol"))
```

(continues on next page)

(continued from previous page)

```
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_protocol = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--timeout"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->timeout = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--other-info"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->other_info = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--mac-address"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->mac_address = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--serial-number"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->serial_number = std::string (argv[i]);
```

(continues on next page)

(continued from previous page)

```
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--file"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->file = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
    }
    if (!board_id_found)
    {
        std::cerr << "board id is not provided" << std::endl;
        return false;
    }
    return true;
}
```

5.5 R

5.5.1 R Get Data from a Board

```
library(brainflow)

params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(brainflow_python$BoardIds$SYNTHETIC_BOARD
↪$value, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 5)
board_shim$stop_stream()
data <- board_shim$get_current_board_data(as.integer(250))
board_shim$release_session()
```

5.5.2 R Get Data from a Board

```
library(brainflow)

params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(brainflow_python$BoardIds$SYNTHETIC_BOARD
↪$value, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 5)
board_shim$insert_marker(1)
board_shim$stop_stream()
data <- board_shim$get_current_board_data(as.integer(250))
board_shim$release_session()
```

5.5.3 R Read Write File

```
library(brainflow)

params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(brainflow_python$BoardIds$SYNTHETIC_BOARD
↪$value, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 5)
board_shim$stop_stream()
data <- board_shim$get_current_board_data(as.integer(250))
board_shim$release_session()

brainflow_python$DataFilter$write_file(data, "test.csv", "w")
data_restored <- brainflow_python$DataFilter$read_file("test.csv")
print(restored_data)
```

5.5.4 R Transforms

```
library(brainflow)

params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(brainflow_python$BoardIds$SYNTHETIC_BOARD
↪$value, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 5)
board_shim$stop_stream()
data <- board_shim$get_current_board_data(as.integer(250))
board_shim$release_session()

# need to convert to numpy array manually
numpy_data <- np$array(data[2,])
print(numpy_data)
wavelet_data <- brainflow_python$DataFilter$perform_wavelet_transform(numpy_data, "db4
↪", as.integer(3))
restored_data <- brainflow_python$DataFilter$perform_inverse_wavelet_
↪transform(wavelet_data, length(numpy_data), "db4", as.integer(3))
```

(continues on next page)

(continued from previous page)

```
print(restored_data)
```

5.5.5 R Signal Filtering

```
library(brainflow)

params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(brainflow_python$BoardIds$SYNTHETIC_BOARD
↪$value, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 5)
board_shim$stop_stream()
data <- board_shim$get_current_board_data(as.integer(250))
board_shim$release_session()

# need to convert to numpy array manually
numpy_data <- np$array(data[2,])
print(numpy_data)
sampling_rate <- board_shim$get_sampling_rate(brainflow_python$BoardIds$SYNTHETIC_
↪BOARD$value)
brainflow_python$DataFilter$perform_bandpass(numpy_data, sampling_rate, 10.0, 5.0, as.
↪integer(3), brainflow_python$FilterTypes$BESSEL$value, 0)
print(numpy_data)
```

5.5.6 R Denoising

```
library(brainflow)

params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(brainflow_python$BoardIds$SYNTHETIC_BOARD
↪$value, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 5)
board_shim$stop_stream()
data <- board_shim$get_current_board_data(as.integer(250))
board_shim$release_session()

# need to convert to numpy array manually
numpy_data <- np$array(data[2,])
print(numpy_data)
brainflow_python$DataFilter$perform_wavelet_denoising(numpy_data, "db4", as.
↪integer(3))
print(numpy_data)
```

5.5.7 R Band Power

```
library(brainflow)

board_id <- brainflow_python$BoardIds$SYNTHETIC_BOARD$value
sampling_rate <- brainflow_python$BoardShim$get_sampling_rate(board_id)
nfft <- brainflow_python$DataFilter$get_nearest_power_of_two(sampling_rate)
params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(board_id, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 10)
board_shim$stop_stream()
data <- board_shim$get_board_data()
board_shim$release_session()

# need to convert to numpy array manually
numpy_data <- np$array(data[3,])
psd <- brainflow_python$DataFilter$get_psd_welch(numpy_data, as.integer(nfft), as.
↪integer(nfft / 2),
    sampling_rate, brainflow_python$WindowFunctions$BLACKMAN_HARRIS$value)
band_power_alpha <- brainflow_python$DataFilter$get_band_power(psd, 7.0, 13.0)
band_power_beta <- brainflow_python$DataFilter$get_band_power(psd, 14.0, 30.0)
ratio <- band_power_alpha / band_power_beta
```

5.5.8 R EEG Metrics

```
library(brainflow)

board_id <- brainflow_python$BoardIds$SYNTHETIC_BOARD$value
sampling_rate <- brainflow_python$BoardShim$get_sampling_rate(board_id)
nfft <- brainflow_python$DataFilter$get_nearest_power_of_two(sampling_rate)
params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(board_id, params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 10)
board_shim$stop_stream()
data <- board_shim$get_board_data()
board_shim$release_session()

eeg_channels <- brainflow_python$BoardShim$get_eeg_channels(board_id)
bands <- brainflow_python$DataFilter$get_avg_band_powers(data, eeg_channels, sampling_
↪rate, TRUE)
feature_vector <- np$array(c(bands[[1]], bands[[2]]))

concentration_params <- brainflow_python$BrainFlowModelParams(brainflow_python
↪$BrainFlowMetrics$CONCENTRATION$value, brainflow_python$BrainFlowClassifiers
↪$REGRESSION$value)
concentration <- brainflow_python$MLModel(concentration_params)
concentration$prepare()
score <- concentration$predict(feature_vector)
concentration$release()
```

5.6 Matlab

5.6.1 Matlab Get Data from a Board

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
board_shim.prepare_session();
a = board_shim.config_board('~6');
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
data = board_shim.get_current_board_data(10);
disp(data);
board_shim.release_session();
```

5.6.2 Matlab Markers

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
board_shim.prepare_session();
a = board_shim.config_board('~6');
board_shim.start_stream(45000, '');
pause(2);
board_shim.insert_marker(1);
pause(2);
board_shim.stop_stream();
data = board_shim.get_board_data();
disp(data);
board_shim.release_session();
```

5.6.3 Matlab Read Write File

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(2)
board_shim.stop_stream()
data = board_shim.get_current_board_data(20);
board_shim.release_session();

DataFilter.write_file(data, 'data.csv', 'w');
restored_data = DataFilter.read_file('data.csv');
```

5.6.4 Matlab Transforms

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
sampling_rate = BoardShim.get_sampling_rate(int32(BoardIDs.SYNTHETIC_BOARD));
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
data = board_shim.get_current_board_data(DataFilter.get_nearest_power_of_two(sampling_
↪rate));
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIDs.SYNTHETIC_BOARD));
% wavelet for first eeg channel %
first_eeg_channel = eeg_channels(1);
original_data = data(first_eeg_channel, :);
[wavelet_data, wavelet_lenghts] = DataFilter.perform_wavelet_transform(original_data,
↪'db4', 2);
restored_data = DataFilter.perform_inverse_wavelet_transform(wavelet_data, wavelet_
↪lenghts, size(original_data, 2), 'db4', 2);
% fft for first eeg channel %
fft_data = DataFilter.perform_fft(original_data, int32(WindowFunctions.NO_WINDOW));
restored_fft_data = DataFilter.perform_ifft(fft_data);
```

5.6.5 Matlab Signal Filtering

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
data = board_shim.get_current_board_data(64);
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIDs.SYNTHETIC_BOARD));
% apply iir filter to the first eeg channel %
first_eeg_channel = eeg_channels(1);
original_data = data(first_eeg_channel, :);
sampling_rate = BoardShim.get_sampling_rate(int32(BoardIDs.SYNTHETIC_BOARD));
filtered_data = DataFilter.perform_lowpass(original_data, sampling_rate, 10.0, 3,␣
↪int32(FilterTypes.BUTTERWORTH), 0.0);
```

5.6.6 Matlab Denoising

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
data = board_shim.get_current_board_data(64);
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIDs.SYNTHETIC_BOARD));
% apply wavelet denoising to the first eeg channel %
first_eeg_channel = eeg_channels(1);
noisy_data = data(first_eeg_channel, :);
denoised_data = DataFilter.perform_wavelet_denoising(noisy_data, 'db4', 2);
```

5.6.7 Matlab Band Power

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
board_id = int32(BoardIDs.SYNTHETIC_BOARD);
board_descr = BoardShim.get_board_descr(board_id);
sampling_rate = int32(board_descr.sampling_rate);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(10);
board_shim.stop_stream();
nfft = DataFilter.get_nearest_power_of_two(sampling_rate);
data = board_shim.get_board_data();
board_shim.release_session();

eeg_channels = board_descr.eeg_channels;
eeg_channel = eeg_channels(3);
original_data = data(eeg_channel, :);
detrended = DataFilter.detrend(original_data, int32(DetrendOperations.LINEAR));
[ampls, freqs] = DataFilter.get_psd_welch(detrended, nfft, nfft / 2, sampling_rate,␣
↪int32(WindowFunctions.HANNING));
band_power_alpha = DataFilter.get_band_power(ampls, freqs, 7.0, 13.0);
band_power_beta = DataFilter.get_band_power(ampls, freqs, 14.0, 30.0);
ratio = band_power_alpha / band_power_beta;
```

5.6.8 Matlab EEG Metrics

```
BoardShim.set_log_file('brainflow.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIDs.SYNTHETIC_BOARD), params);
sampling_rate = BoardShim.get_sampling_rate(int32(BoardIDs.SYNTHETIC_BOARD));
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
nfft = DataFilter.get_nearest_power_of_two(sampling_rate);
data = board_shim.get_board_data();
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIDs.SYNTHETIC_BOARD));
[avgs, stddevs] = DataFilter.get_avg_band_powers(data, eeg_channels, sampling_rate,␣
↪true);
feature_vector = double([avgs, stddevs]);

concentration_params = BrainFlowModelParams(int32(BrainFlowMetrics.CONCENTRATION),␣
↪int32(BrainFlowClassifiers.REGRESSION));
concentration = MLModel(concentration_params);
concentration.prepare();
score = concentration.predict(feature_vector);
concentration.release();
```

5.7 Julia

5.7.1 Julia Get Data from a Board

```
using BrainFlow

# specify logging library to use
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(5)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_current_board_data(256, board_shim)
BrainFlow.release_session(board_shim)
```

5.7.2 Julia Markers

```
using BrainFlow

BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim, 45000, "file://data.csv:w")
sleep(1)
BrainFlow.insert_marker(1.0, board_shim)
sleep(1)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_current_board_data(256, board_shim)
BrainFlow.release_session(board_shim)
```

5.7.3 Julia Read Write File

```
using BrainFlow


# specify logging library to use
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(5)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_current_board_data(32, board_shim)
BrainFlow.release_session(board_shim)

BrainFlow.write_file(data, "test.csv", "w")
restored_data = BrainFlow.read_file("test.csv")

println("Original Data")
println(data)
println("Restored Data")
println(restored_data)
```

5.7.4 Julia Transforms

```
using BrainFlow

# enable logs
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)
BrainFlow.enable_dev_logger(BrainFlow.DATA_HANDLER)

params = BrainFlowInputParams()
```

(continues on next page)

(continued from previous page)

```
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)
sampling_rate = BrainFlow.get_sampling_rate(BrainFlow.SYNTHETIC_BOARD)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(10)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_current_board_data(BrainFlow.get_nearest_power_of_two(sampling_
↪rate), board_shim)
BrainFlow.release_session(board_shim)

eeg_channels = BrainFlow.get_eeg_channels(BrainFlow.SYNTHETIC_BOARD)
data_first_channel = data[eeg_channels[1], :]

# returns tuple of wavelet coeffs and lengths
wavelet_data = BrainFlow.perform_wavelet_transform(data_first_channel, "db4", 2)
restored_wavelet_data = BrainFlow.perform_inverse_wavelet_transform(wavelet_data,␣
↪length(data_first_channel), "db4", 2)

fft_data = BrainFlow.perform_fft(data_first_channel, BrainFlow.NO_WINDOW)
restored_fft_data = BrainFlow.perform_ifft(fft_data)

println("Original Data")
println(data_first_channel)
println("Restored from Wavelet Data")
println(restored_wavelet_data)
println("Restored from FFT Data")
println(restored_fft_data)
```

5.7.5 Julia Signal Filtering

```
using BrainFlow


# specify logging library to use
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(5)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_current_board_data(32, board_shim)
BrainFlow.release_session(board_shim)

eeg_channels = BrainFlow.get_eeg_channels(BrainFlow.SYNTHETIC_BOARD)
sampling_rate = BrainFlow.get_sampling_rate(BrainFlow.SYNTHETIC_BOARD)

data_first_channel = data[eeg_channels[1], :]
println("Original Data First Channel")
println(data_first_channel)
BrainFlow.perform_lowpass(data_first_channel, sampling_rate, 10.0, 3, BrainFlow.
↪BUTTERWORTH, 0.0)
```

(continues on next page)

(continued from previous page)

```
println("After LowPass Filter")
println(data_first_channel)

data_second_channel = data[eeg_channels[2], :]
println("Original Data Second Channel")
println(data_second_channel)
BrainFlow.perform_highpass(data_second_channel, sampling_rate, 5.0, 3, BrainFlow.
↪CHEBYSHEV_TYPE_1, 1.0)
println("After HighPass Filter")
println(data_second_channel)

data_third_channel = data[eeg_channels[3], :]
println("Original Data Third Channel")
println(data_third_channel)
BrainFlow.perform_bandpass(data_third_channel, sampling_rate, 25.0, 20.0, 3,␣
↪BrainFlow.BESSEL, 0.0)
println("After BandPass Filter")
println(data_third_channel)

data_fourth_channel = data[eeg_channels[4], :]
println("Original Data Fourth Channel")
println(data_fourth_channel)
BrainFlow.perform_bandstop(data_fourth_channel, sampling_rate, 50.0, 2.0, 3,␣
↪BrainFlow.BESSEL, 0.0)
println("After BandStop Filter")
println(data_fourth_channel)

data_fifth_channel = data[eeg_channels[5], :]
println("Original Data Fifth Channel")
println(data_fifth_channel)
BrainFlow.remove_environmental_noise(data_fifth_channel, sampling_rate, BrainFlow.
↪FIFTY)
println("After BandStop Filter")
println(data_fifth_channel)
```

5.7.6 Julia Denoising

```
using BrainFlow

# specify logging library to use
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(5)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_current_board_data(32, board_shim)
BrainFlow.release_session(board_shim)

eeg_channels = BrainFlow.get_eeg_channels(BrainFlow.SYNTHETIC_BOARD)
sampling_rate = BrainFlow.get_sampling_rate(BrainFlow.SYNTHETIC_BOARD)
```

(continues on next page)

(continued from previous page)

```
data_first_channel = data[eeg_channels[1], :]
println("Original Data First Channel")
println(data_first_channel)
BrainFlow.perform_rolling_filter(data_first_channel, 3, BrainFlow.MEAN)
println("After Rolling Filter")
println(data_first_channel)

data_second_channel = data[eeg_channels[2], :]
println("Original Data Second Channel")
println(data_second_channel)
BrainFlow.perform_wavelet_denoising(data_second_channel, "db4", 2)
println("After Wavelet Denoising")
println(data_second_channel)
```

5.7.7 Julia Band Power

```
using BrainFlow

# specify logging library to use
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)
board_descr = BrainFlow.get_board_descr(BrainFlow.SYNTHETIC_BOARD)
sampling_rate = board_descr["sampling_rate"]
nfft = BrainFlow.get_nearest_power_of_two(sampling_rate)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(5)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_board_data(board_shim)
BrainFlow.release_session(board_shim)

eeg_channels = board_descr["eeg_channels"]
# second channel of synthetic board is sine wave at 10 Hz, should see huge 'alpha'
data_second_channel = data[eeg_channels[2], :]

# optional: detrend
BrainFlow.detrend(data_second_channel, BrainFlow.LINEAR)
# psd is a tuple of ampls and freqs
psd = BrainFlow.get_psd_welch(data_second_channel, nfft, Integer(nfft / 2), sampling_
↪rate, BrainFlow.BLACKMAN_HARRIS)
band_power_alpha = BrainFlow.get_band_power(psd, 7.0, 13.0)
band_power_beta = BrainFlow.get_band_power(psd, 14.0, 30.0)
println(band_power_alpha / band_power_beta)
```

5.7.8 Julia EEG Metrics

```
using BrainFlow

# enable all possible logs from all three libs
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)
BrainFlow.enable_dev_logger(BrainFlow.DATA_HANDLER)
BrainFlow.enable_dev_logger(BrainFlow.ML_MODULE)

params = BrainFlowInputParams()
board_shim = BrainFlow.BoardShim(BrainFlow.SYNTHETIC_BOARD, params)
sampling_rate = BrainFlow.get_sampling_rate(BrainFlow.SYNTHETIC_BOARD)
nfft = BrainFlow.get_nearest_power_of_two(sampling_rate)

BrainFlow.prepare_session(board_shim)
BrainFlow.start_stream(board_shim)
sleep(5)
BrainFlow.stop_stream(board_shim)
data = BrainFlow.get_board_data(board_shim)
BrainFlow.release_session(board_shim)

eeg_channels = BrainFlow.get_eeg_channels(BrainFlow.SYNTHETIC_BOARD)

bands = BrainFlow.get_avg_band_powers(data, eeg_channels, sampling_rate, true)
feature_vector = vcat(bands[1], bands[2])

# calc concentration
model_params = BrainFlowModelParams(metric = "concentration", classifier = "KNN")
BrainFlow.prepare(model_params)
print(BrainFlow.predict(feature_vector, model_params))
BrainFlow.release(model_params)

# calc relaxation
model_params = BrainFlowModelParams(metric = "relaxation", classifier = "regression")
BrainFlow.prepare(model_params)
print(BrainFlow.predict(feature_vector, model_params))
BrainFlow.release(model_params)
```

5.8 Notebooks

5.8.1 BrainFlow to MNE Python Notebook

In [1]:
```
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

import brainflow
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds

import mne
from mne.channels import read_layout
```

```
/home/docs/checkouts/readthedocs.org/user_builds/brainflow/envs/master/lib/python3.7/
↪site-packages/traitlets/traitlets.py:3036: FutureWarning: --rc={'figure.dpi': 96}␣
↪for dict-traits is deprecated in traitlets 5.0. You can pass --rc <key=value> ...␣
↪multiple times to add items to a dict.
  FutureWarning,
```

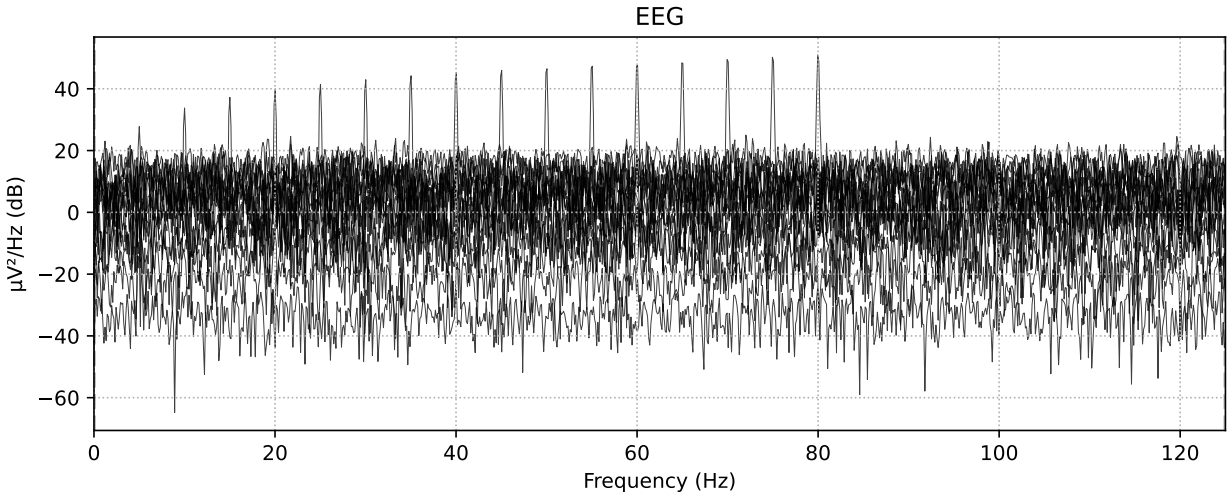
In [2]:
```
# use synthetic board for demo
params = BrainFlowInputParams()
board = BoardShim(BoardIds.SYNTHETIC_BOARD.value, params)
board.prepare_session()
board.start_stream()
time.sleep(10)
data = board.get_board_data()
board.stop_stream()
board.release_session()
```

In [3]:
```
eeg_channels = BoardShim.get_eeg_channels(BoardIds.SYNTHETIC_BOARD.value)
eeg_data = data[eeg_channels, :]
eeg_data = eeg_data / 1000000 # BrainFlow returns uV, convert to V for MNE
```

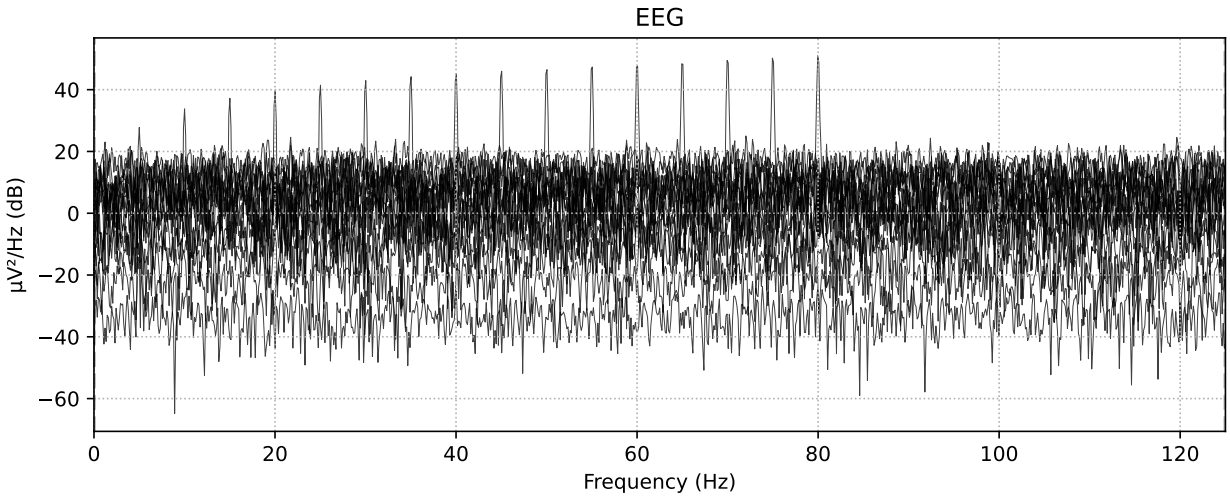
In [4]:
```
# Creating MNE objects from brainflow data arrays
ch_types = ['eeg'] * len(eeg_channels)
ch_names = BoardShim.get_eeg_names(BoardIds.SYNTHETIC_BOARD.value)
sfreq = BoardShim.get_sampling_rate(BoardIds.SYNTHETIC_BOARD.value)
info = mne.create_info(ch_names=ch_names, sfreq=sfreq, ch_types=ch_types)
raw = mne.io.RawArray(eeg_data, info)
# its time to plot something!
raw.plot_psd(average=False)
```

```
Creating RawArray with float64 data, n_channels=16, n_times=2482
    Range : 0 ... 2481 =      0.000 ...     9.924 secs
Ready.
Effective window size : 8.192 (s)
```

```
<ipython-input-1-26921b74558c>:8: RuntimeWarning: Channel locations not available.␣
↪Disabling spatial colors.
  raw.plot_psd(average=False)
```



Out[4]:



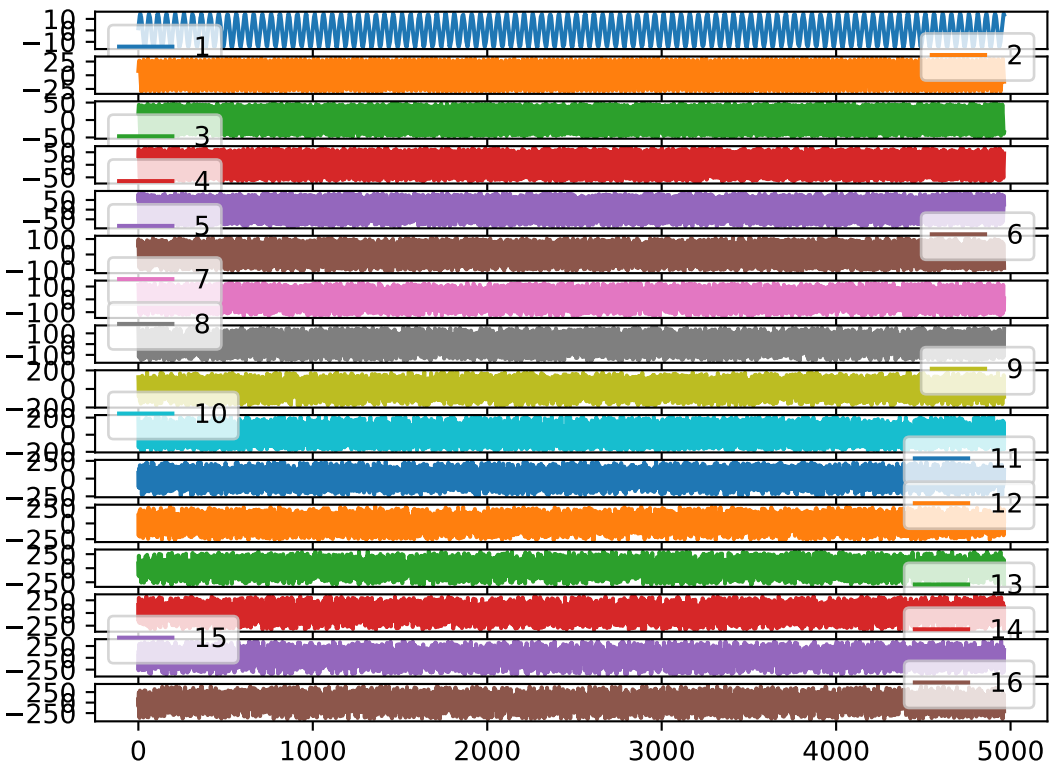
5.8.2 Denoising Notebook

In [1]:
```
import argparse
import time
import brainflow
import numpy as np

import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations
```

```
/home/docs/checkouts/readthedocs.org/user_builds/brainflow/envs/master/lib/python3.7/
↪site-packages/traitlets/traitlets.py:3036: FutureWarning: --rc={'figure.dpi': 96}␣
↪for dict-traits is deprecated in traitlets 5.0. You can pass --rc <key=value> ...␣
↪multiple times to add items to a dict.
  FutureWarning,
```

In [2]:
```
# use synthetic board for demo
params = BrainFlowInputParams()
board_id = BoardIds.SYNTHETIC_BOARD.value
board = BoardShim(board_id, params)
board.prepare_session()
board.start_stream()
time.sleep(20)
data = board.get_board_data()
board.stop_stream()
board.release_session()
```

In [3]:
```
# plot original data
eeg_channels = BoardShim.get_eeg_channels(board_id)
df = pd.DataFrame(np.transpose(data))
df[eeg_channels].plot(subplots=True)
plt.show()
```



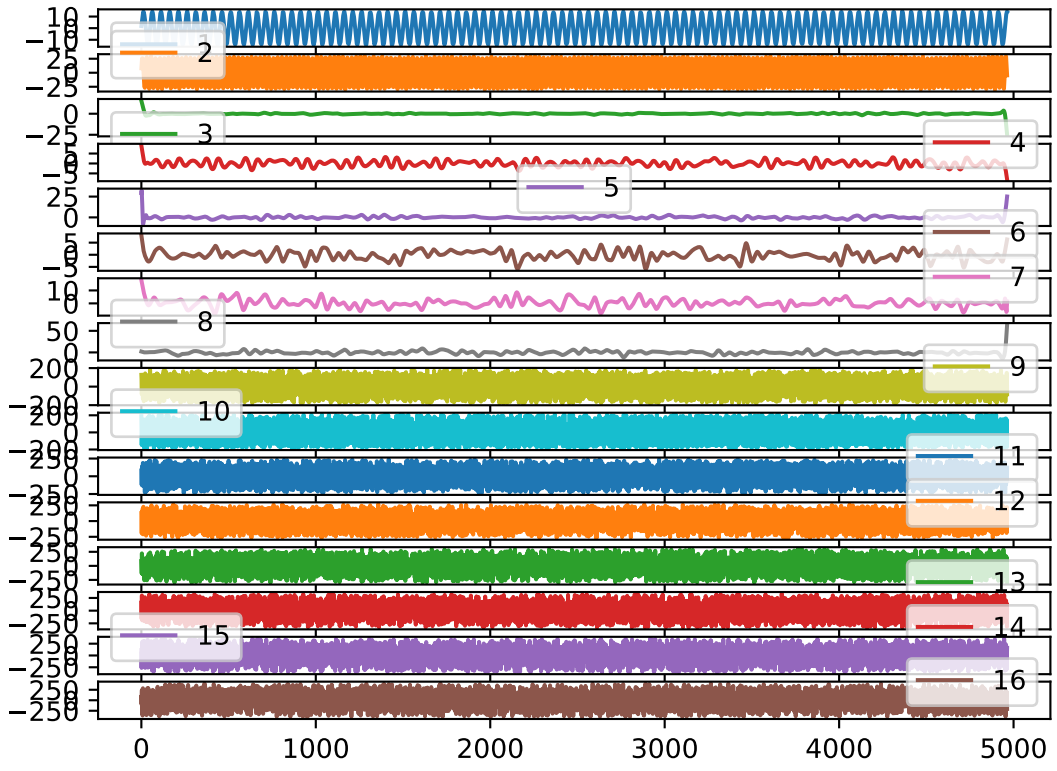
In [4]:
```
# demo for different denoising methods,
# apply different methods to different channels to determine the best one
for count, channel in enumerate(eeg_channels):
    # first of all you can try simple moving median or moving average with different␣
↪window size
    if count == 0:
        DataFilter.perform_rolling_filter(data[channel], 3, AggOperations.MEAN.value)
    elif count == 1:
        DataFilter.perform_rolling_filter(data[channel], 3, AggOperations.MEDIAN.
↪value)
    # methods above should increase signal to noise ratio but we can do even better
    # using wavelet based denoising, feel free to try different wavelet functions and␣
↪decomposition levels
    elif count == 2:
        DataFilter.perform_wavelet_denoising(data[channel], 'db6', 5)
    elif count == 3:
        DataFilter.perform_wavelet_denoising(data[channel], 'bior3.9', 5)
    elif count == 4:
        DataFilter.perform_wavelet_denoising(data[channel], 'sym7', 5)
    elif count == 5:
        DataFilter.perform_wavelet_denoising(data[channel], 'coif3', 5)
    elif count == 6:
        DataFilter.perform_wavelet_denoising(data[channel], 'bior6.8', 5)
    elif count == 7:
        DataFilter.perform_wavelet_denoising(data[channel], 'db4', 5)
```

In [5]:
```
# plot denoised data
df = pd.DataFrame(np.transpose(data))
```

(continues on next page)

(continued from previous page)

```
df[eeg_channels].plot(subplots=True)
plt.show()
# as you can see wavelet based denoising works much better and increases signal to␣
↪noise ratio significantly!
```



5.8.3 BrainFlow Band Power Notebook

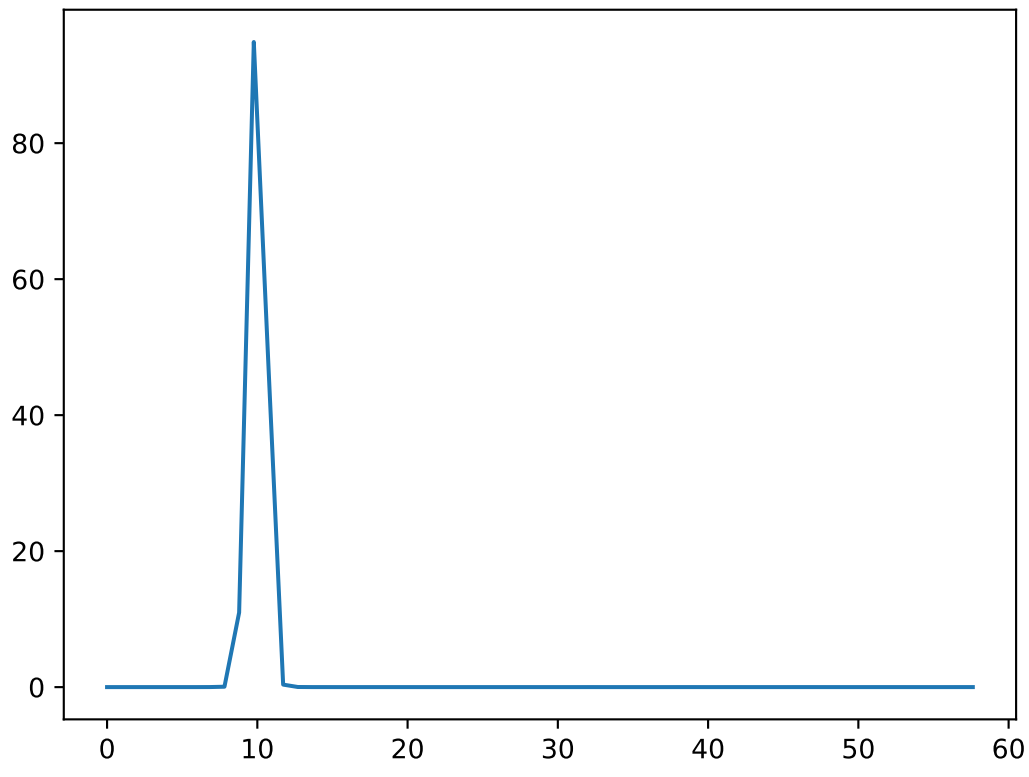
In [1]:
```
import argparse
import time
import brainflow
import numpy as np

import pandas as pd
import matplotlib
import matplotlib.pyplot as plt

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, AggOperations,␣
↪WindowFunctions, DetrendOperations
```

```
/home/docs/checkouts/readthedocs.org/user_builds/brainflow/envs/master/lib/python3.7/
↪site-packages/traitlets/traitlets.py:3036: FutureWarning: --rc={'figure.dpi': 96}␣
↪for dict-traits is deprecated in traitlets 5.0. You can pass --rc <key=value> ...␣
↪multiple times to add items to a dict.
  FutureWarning,
```

In [2]:
```
# use synthetic board for demo
params = BrainFlowInputParams()
board_id = BoardIds.SYNTHETIC_BOARD.value
sampling_rate = BoardShim.get_sampling_rate(board_id)
nfft = DataFilter.get_nearest_power_of_two(sampling_rate)
board = BoardShim(board_id, params)
board.prepare_session()
board.start_stream()
time.sleep(10)
data = board.get_board_data()
board.stop_stream()
board.release_session()
eeg_channels = BoardShim.get_eeg_channels(board_id)
# use first eeg channel for demo
# second channel of synthetic board is a sine wave at 10 Hz, should see big 'alpha'
eeg_channel = eeg_channels[1]
```

In [3]:
```
# optional: detrend
DataFilter.detrend(data[eeg_channel], DetrendOperations.LINEAR.value)
```

In [4]:
```
psd = DataFilter.get_psd_welch(data[eeg_channel], nfft, nfft // 2, sampling_rate,␣
↪WindowFunctions.HANNING.value)
plt.plot(psd[1][:60], psd[0][:60])
plt.show()
```



In [5]:
```
# calc band power
alpha = DataFilter.get_band_power(psd, 7.0, 13.0)
beta = DataFilter.get_band_power(psd, 14.0, 30.0)
print("Alpha/Beta Ratio is: %f" % (alpha / beta))
```

```
Alpha/Beta Ratio is: 2573.115178
```

CHAPTER

SIX

INTEGRATION WITH GAME ENGINES

6.1 Unity

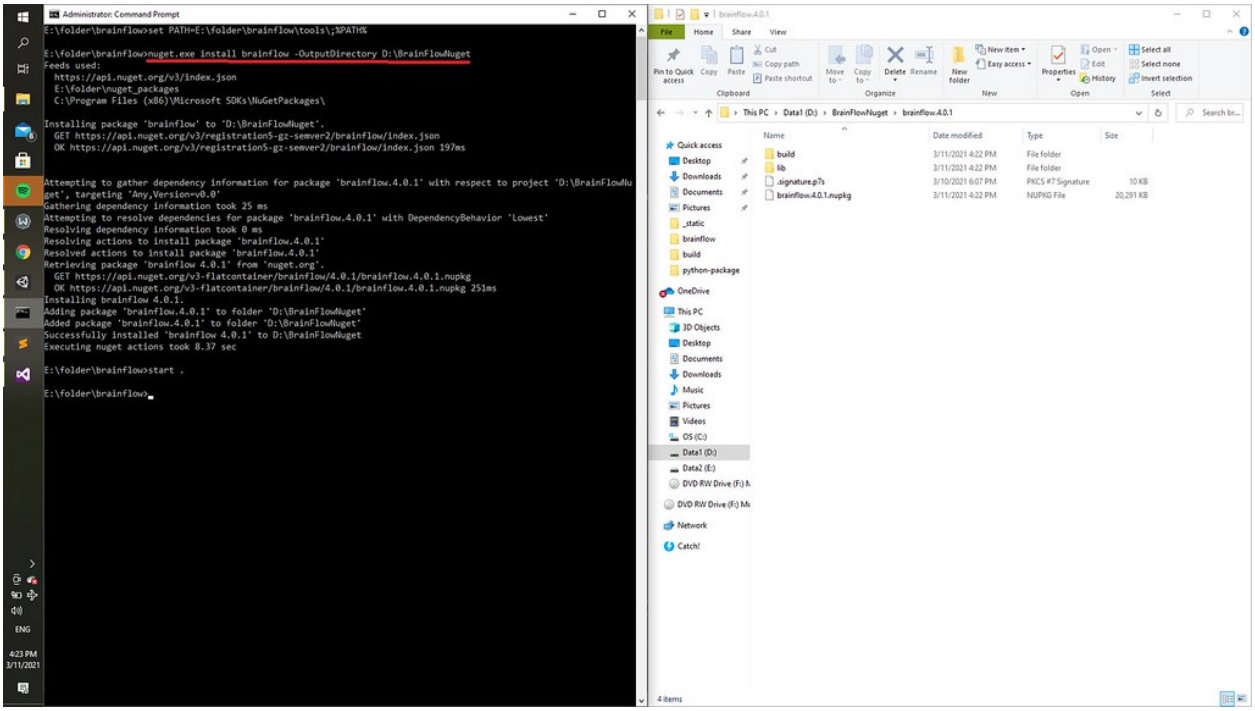
Integration with Unity can be done only using C# binding. We tested it only on Windows, but it may work on Linux and MacOS too. Android and IOS are not supported.

You can build C# binding from source or download compiled package directly from Nuget.

Here we will use Nuget to download and install BrainFlow.

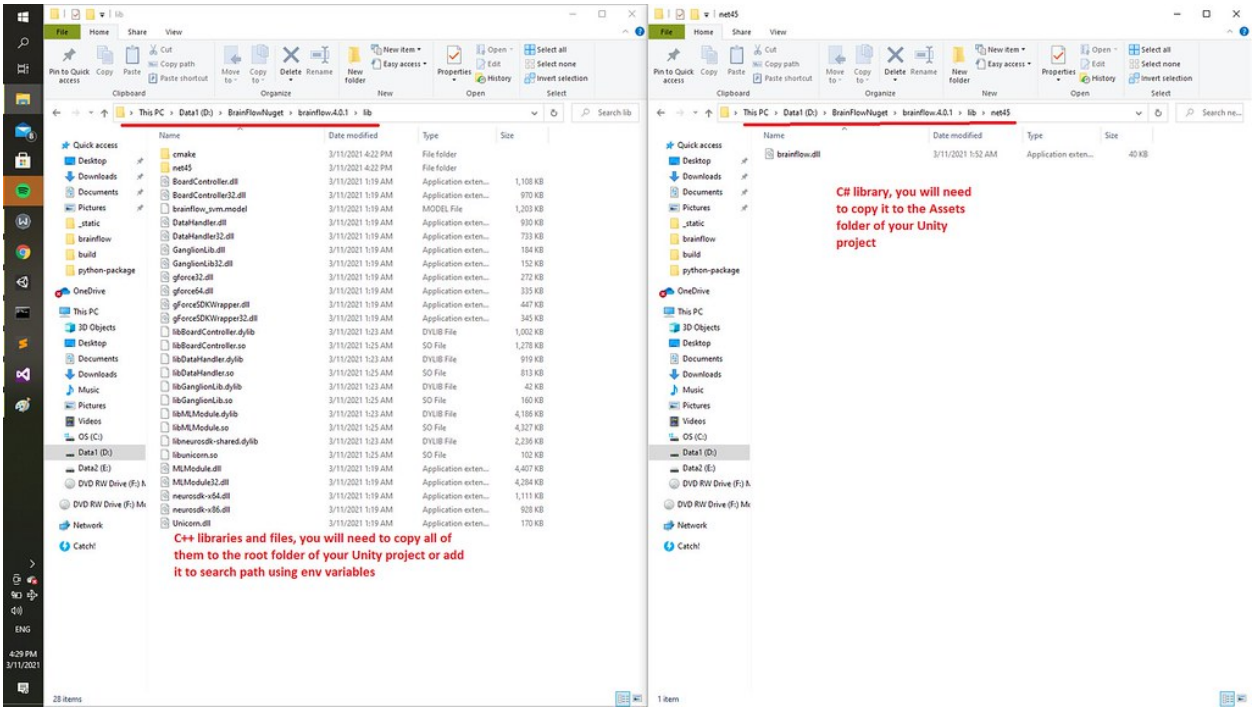
Download nuget.exe and run

```
nuget.exe install brainflow -OutputDirectory <OUTPUTDIR>
```

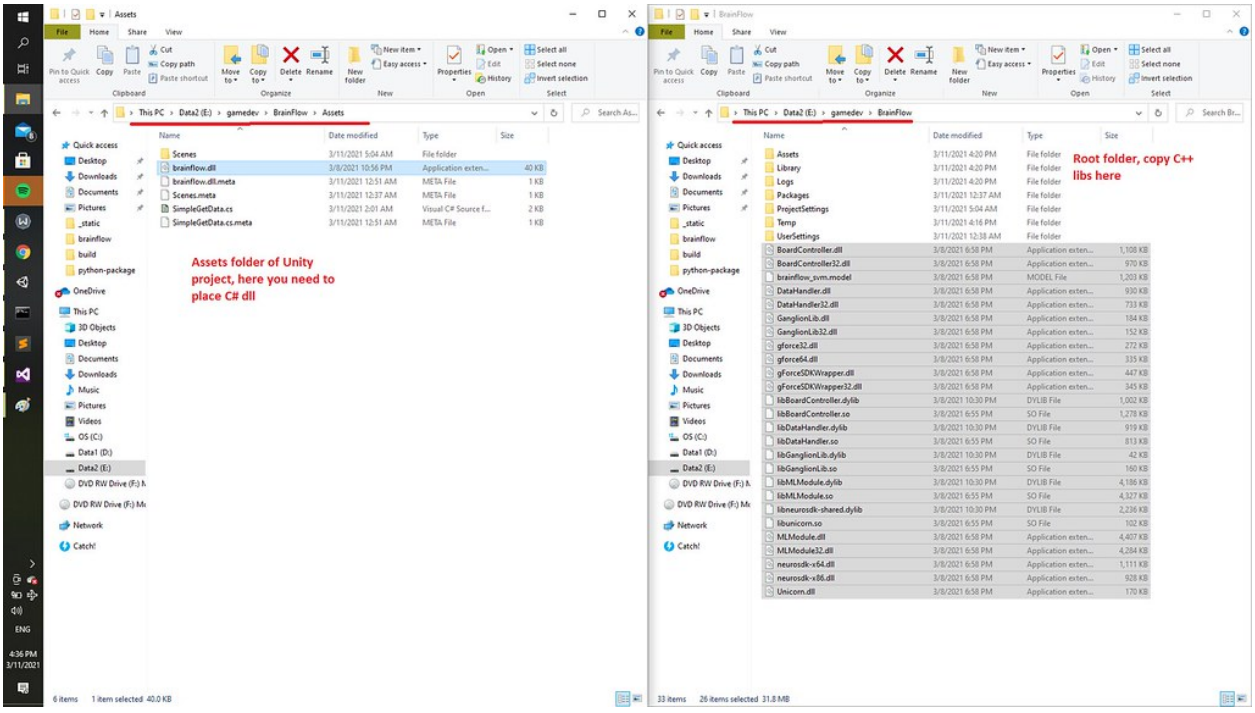


Open OUTPUTDIR, in our example it is *D:\BrainFlowNuget*. At the moment of writing this tutorial latest BrainFlow version is 4.0.1, it is ok if you download newer version from Nuget, it does not affect the process of integration with Unity.

For BrainFlow there are *Managed(C#)* and *Unmanaged(C++)* libraries. C++ libraries are located inside folder *D:\BrainFlowNuget\brainflow.4.0.1\lib*, C# libraries are located inside folder *D:\BrainFlowNuget\brainflow.4.0.1\lib\net45*.



Open your Unity project and copy **Managed(C#)** libraries to the Assets folder, after that copy **Unmanaged(C++)** libraries to the root folder of your project.



Now, you are able to use BrainFlow API in your Unity project.

For demo we will create a simple script to read data.

Add a game object to the Scene and attach script below.

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

using brainflow;
using brainflow.math;

public class SimpleGetData : MonoBehaviour
{
    private BoardShim board_shim = null;
    private int sampling_rate = 0;

    // Start is called before the first frame update
    void Start()
    {
        try
        {
            BoardShim.set_log_file("brainflow_log.txt");
            BoardShim.enable_dev_board_logger();

            BrainFlowInputParams input_params = new BrainFlowInputParams();
            int board_id = (int)BoardIds.SYNTHETIC_BOARD;
            board_shim = new BoardShim(board_id, input_params);
            board_shim.prepare_session();
            board_shim.start_stream(450000, "file://brainflow_data.csv:w");
            sampling_rate = BoardShim.get_sampling_rate(board_id);
            Debug.Log("Brainflow streaming was started");
        }
        catch (BrainFlowException e)
        {
            Debug.Log(e);
        }
    }

    // Update is called once per frame
    void Update()
    {
        if (board_shim == null)
        {
            return;
        }
        int number_of_data_points = sampling_rate * 4;
        double[,] data = board_shim.get_current_board_data(number_of_data_points);
        // check https://brainflow.readthedocs.io/en/stable/index.html for api ref␣
↪and more code samples
        Debug.Log("Num elements: " + data.GetLength(1));
    }

    // you need to call release_session and ensure that all resources correctly␣
↪released
    private void OnDestroy()
    {
        if (board_shim != null)
        {
            try
            {
```

(continues on next page)

(continued from previous page)

```
                board_shim.release_session();
            }
            catch (BrainFlowException e)
            {
                Debug.Log(e);
            }
            Debug.Log("Brainflow streaming was stopped");
        }
    }
}
```

After building your game for production don't forget to copy *Unmanaged(C++)* libraries to a folder where executable is located.

6.2 Unreal Engine

We provide Unreal Engine Plugin with precompiled libraries for most commonly used configurations. Check Readme for installtion details.

This blog post can help if you want to write your own plugin or extend existing one.

6.3 CryEngine

CryEngine uses CMake, build BrainFlow by yourself first and check C++ examples for instructions to integrate BrainFlow into CMake projects.

Keep in mind MSVC runtime linking, default in BrainFlow is static, you can provide `-DMSVC_RUNTIME=dynamic` or `-DMSVC_RUNTIME=static` to control it.

CHAPTER

SEVEN

BRAINFLOW DEV

7.1 Code style

We use clang-format tool to keep the same code style for all cpp files. You can download clang-format binary from LLVM Download Page We recommend installing a plugin for your text editor or IDE which will apply clang-format tool during saving. You will need to set code style option to "FILE"

Plugins for text editors and IDEs:

- Sublime
- VSCode
- Guide for Visual Studio

Unfortunately clang-format cannot handle naming, so some additional rules are:

- methods and variables should be in lower case with underscore
- class names should be in camel case
- use brackets even for single line if and for statements

For C# we use the same code style as for C++, for java there is a formatter file to take care of code style.

7.2 CI and tests

If you want to commit to the core module of BrainFlow project please check that all tests are passed, you should check CI status in your PR and fix all errors if any. Also, you are able to run failed tests locally using BrainFlow emulator.

In CI warnings as errors option is enabled for C++ code and you need to fix all of them. Also, we have CppCheck static analysis tool. If you see that such check failed you need to download artifact from CppCheck Github Acttion, open generated html report and fix errors.

7.3 Pull Requests

Just try to briefly explain a goal of this PR.

7.4 Instructions to add new boards to BrainFlow

- add new board Id to BoardIds enum in C code and to the same enum in all bindings
- add new object creation to board controller C interface
- inherit your board from Board class and implement all pure virtual methods, store data in DataBuffer object, use synthetic board as a reference, try to reuse code from utils folder
- add information about your board to brainflow_boards.cpp
- add new files to BOARD_CONTROLLER_SRC variable in CmakeLists.txt, you may also need to add new directory to *target_include_directories* for BOARD_CONTROLLER_NAME variable

You've just written Python, Java, C#, R, C++ ... SDKs for your board! Also, now you can use your new board with applications and frameworks built on top of BrainFlow API.

Optional: We use CI to run tests automatically, to add your board to CI pipelines you can develop a simple emulator for your device. Use emulators for existing boards as a reference and add tests for your device to Github Acttions workflows.

7.5 Instructions to build docs locally

Don't push changes to Docs without local verification.

- install pandoc
- optional: install Doxygen, skip it if you dont understand what it is or don't need to publish your local build

Install requirements:

```
cd docs
python -m pip install -r requirements.txt
```

Build docs:

```
make html
```

7.6 Debug BrainFlow's errors

Since bindings just call methods from dynamic libraries, more likely errors occur in C++ code, it means that you need to use C++ debuger like gdb. If there is an error in binding, it should be simple to figure out and resolve the issue using language specific tools.

Steps to get more information about errors in C++ code:

- build BrainFlow's core module and C++ binding in debug mode. In files like tools/build_linux.sh default config is Release, so you need to change it to Debug
- reproduce your issue using C++ binding

- run it with debuger and memory checker

Example for Linux(for MacOS it's the same):

```
vim tools/build_linux.sh
# Change build type to Debug
bash tools/build_linux.sh
# Create a test to reproduce your issue in C++, here we will use get_data_demo
cd tests/cpp/get_data_demo
mkdir build
cd build
cmake -DCMAKE_PREFIX_PATH=TYPE_FULL_PATH_TO_BRAINFLOW_INSTALLED_FOLDER -DCMAKE_BUILD_
↪TYPE=Debug ..
# e.g. cmake -DCMAKE_PREFIX_PATH=/home/andrey/brainflow/installed_linux -DCMAKE_BUILD_
↪TYPE=Debug ..
make
# Run Valgrind to check memory errors
# Here we use command line for Ganglion
sudo valgrind --error-exitcode=1 --leak-check=full ./brainflow_get_data --board-id 1 -
↪-serial-port /dev/ttyACM0 --mac-address e6:73:73:18:09:b1
# Valgrind will print Error Summary and exact line numbers
# Run gdb and get backtrace
sudo gdb --args ./brainflow_get_data --board-id 1 --serial-port /dev/ttyACM0 --mac-
↪address e6:73:73:18:09:b1
# In gdb terminal type 'r' to run the program and as soon as error occurs, type 'bt'␣
↪to see backtrace with exact lines of code and call stack
```

7.7 BrainFlow Emulator

BrainFlow Emulator allows you to run all integration tests for all supported boards without real hardware. Our CI uses it for test automation. Also, you can run it on your own PC!

Emulators listed here intended for CI and run process to test automatically. It's great for automation but not easy to use if you need to test it in GUI. So, for some devices there are manual emulators as well. Such emulators don't run process to test for you. Manual emulators make it easier to run tests in GUI based applications.

7.7.1 Streaming Board

Streaming Board emulator works using Python binding for BrainFlow, so **you need to install Python binding first.**

Install emulator package:

```
cd emulator
python -m pip install -U .
```

Run tests

```
python emulator\brainflow_emulator\streaming_board_emulator.py python tests\python\
↪brainflow_get_data.py --log --board-id -2 --ip-address 225.1.1.1 --ip-port 6677 --
↪other-info -1
```

This emulator uses synthetic board as a master board and, IP address and port are hardcoded.

7.7.2 OpenBCI Cyton

Cyton emulator simulate COM port using:

- com0com for Windows
- pty for Linux and MacOS

You should pass test command line directly to cyton_linux.py or to cyton_windows.py. The script will add the port automatically to provided command line and will start an application.

Install emulator package:

```
cd emulator
python -m pip install -U .
```

Run tests for LinuxMacOS and Windows (port argument will be added by Emulator!)

```
python brainflow_emulator/cyton_linux.py python ../tests/python/brainflow_get_data.py␣
↪--log --board-id 0 --serial-port
python brainflow_emulator\cyton_windows.py python ..\tests\python\brainflow_get_data.
↪py --log --board-id 0 --serial-port
```

7.7.3 Galea

Galea emulator creates socket server and streams data to BrainFlow like it's a real board.

Install emulator package:

```
cd emulator
python -m pip install -U .
```

Run tests:

```
python brainflow_emulator/galea_udp.py python ../tests/python/brainflow_get_data.py --
↪log --ip-address 127.0.0.1 --board-id 3
```

7.7.4 OpenBCI Wifi Shield based boards

Wifi shield emulator starts http server to read commands and creates client socket to stream data.

Install emulator package:

```
cd emulator
python -m pip install -U .
```

Run tests for Ganglion, Cyton and Daisy with Wifi Shield:

```
python brainflow_emulator/wifi_shield_emulator.py python ../tests/python/brainflow_
↪get_data.py --log --ip-address 127.0.0.1 --board-id 4 --ip-protocol 2 --ip-port␣
↪17982
python brainflow_emulator/wifi_shield_emulator.py python ../tests/python/brainflow_
↪get_data.py --log --ip-address 127.0.0.1 --board-id 5 --ip-protocol 2 --ip-port␣
↪17982
python brainflow_emulator/wifi_shield_emulator.py python ../tests/python/brainflow_
↪get_data.py --log --ip-address 127.0.0.1 --board-id 6 --ip-protocol 2 --ip-port␣
↪17982
```

7.7.5 FreeEEG32

FreeEEG32 emulator simulate COM port using:

- com0com for Windows
- pty for Linux and MacOS

You should pass test command line directly to freeeeg32_linux.py or to freeeeg32_windows.py. The script will add the port automatically to provided command line and will start an application.

Install emulator package:

```
cd emulator
python -m pip install -U .
```

Run tests for LinuxMacOS and Windows (port argument will be added by Emulator!)

```
python brainflow_emulator/freeeeg32_linux.py python ../tests/python/brainflow_get_
↪data.py --log --board-id 17 --serial-port
python brainflow_emulator\freeeeg32_windows.py python ..\tests\python\brainflow_get_
↪data.py --log --board-id 17 --serial-port
```

CHAPTER

EIGHT

ASK HELP

8.1 Contact Info, Feature Request, Report an Issue

- Join our slack workspace using self-invite page
- To report bugs or request features create an issue in our GitHub Page
- For hardware related questions and issues contact board manufacturer

8.2 Issue format

First of all you need to run your code with:

```
enable_dev_board_logger ()
```

After that, make sure:

- you've specified BrainFlow version and OS version
- you've attached all logs to your issue description
- you've provided steps or a simple example to reproduce your issue

8.3 Contributors

- Andrey1994 - 843 contributions
- daniellasry - 26 contributions
- xloem - 8 contributions
- matthijscox - 6 contributions
- shirleyzhang867 - 5 contributions
- mesca - 5 contributions
- Fan1117 - 5 contributions
- John42506176Linux - 4 contributions
- imachug - 3 contributions
- retiutut - 3 contributions

CHAPTER

NINE

PARTNERS AND SPONSORS

9.1 OpenBCI

OpenBCI specializes in creating low-cost, high-quality biosensing hardware for brain computer interfacing. Their arduino compatible biosensing boards provide high resolution imaging and recording of EMG, ECG, and EEG signals. Their devices have been used by researchers, makers, and hobbyists in over 60+ countries as brain computer interfaces to power machines and map brain activity. OpenBCI headsets, boards, sensors and electrodes allow anyone interested in biosensing and neurofeedback to purchase high quality equipment at affordable prices.

CHAPTER

TEN

MIT LICENSE

CHAPTER

ELEVEN

PARTNERS AND SPONSORS

MATLAB MODULE INDEX

b

INDEX

C

D

E

F

G

I

L

M

N

P

R

S

T

W