# BrainFlow Documentation

**Andrey Parfenov**

**Apr 23, 2024**

# 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

## 1.1 BrainFlow Dummy Boards

### 1.1.1 Playback File Board

This board playbacks files recorded using other BrainFlow boards.

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

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.PLAYBACK_FILE_BOARD`
- `master_board`, it should contain board ID of the device used to create playback files
- `file`, it should contain full path to recorded file
- *optional:* `file_aux`, use it if your master board has auxiliary preset
- *optional:* `file_anc`, use it if your master board has ancillary preset

Initialization Example:

```
params = BrainFlowInputParams()
params.file = "streamer_default.csv"
params.file_aux = "streamer_aux.csv"
params.master_board = BoardIds.SYNTHETIC_BOARD
board = BoardShim(BoardIds.PLAYBACK_FILE_BOARD, params)
```

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Devices like Raspberry Pi

By default it generates new timestamps and stops at the end of the file. You can override such behavior 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.

### 1.1.2 Streaming Board

BrainFlow boards can stream data to different destinations like file, socket, etc directly from BrainFlow. This board acts like a consumer for data streamed from the main process.

To use it in the first process(master process, data provider) you should call:

```
# choose any valid multicast address(from "224.0.0.0" to "239.255.255.255") and port
add_streamer ("streaming_board://225.1.1.1:6677", BrainFlowPresets.DEFAULT_PRESET)
```

In the second process you shoud create Streaming board instance and this process will act as a data consumer.

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.STREAMING_BOARD`
- `ip_address`, for example above it's 225.1.1.1
- `ip_port`, for example above it's 6677
- `master_board`, it should contain board ID of the device used to create playback files
- *optional:* `ip_address_aux`, use it if your master board has auxiliary preset
- *optional:* `ip_port_aux`, use it if your master board has auxiliary preset
- *optional:* `ip_address_anc`, use it if your master board has ancillary preset
- *optional:* `ip_port_anc`, use it if your master board has ancillary preset

Initialization Example:

```
params = BrainFlowInputParams()
params.ip_port = 6677
params.ip_port_aux = 6678
params.ip_address = "225.1.1.1"
params.ip_address_aux = "225.1.1.1"
params.master_board = BoardIds.SYNTHETIC_BOARD
board = BoardShim(BoardIds.STREAMING_BOARD, params)
```

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Devices like Raspberry Pi
- Android

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.

If you have problems on Windows try to disable virtual box network adapter and firewall. More info can be found here.

### 1.1.3 Synthetic Board

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

**It can be extremely useful during development.**

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.SYNTHETIC_BOARD`

Initialization Example:

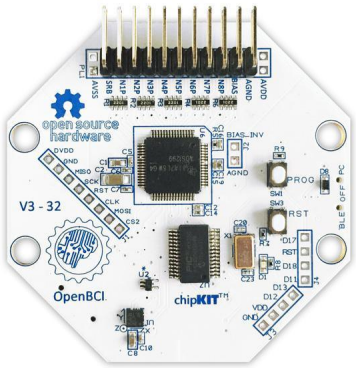
```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.SYNTHETIC_BOARD, params)
```

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Devices like Raspberry Pi
- Android

## 1.2 OpenBCI

### 1.2.1 Cyton




Cyton Getting Started Guide from OpenBCI

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.CYTON_BOARD`
- `serial_port`, e.g. COM3, /dev/ttyUSB0, etc

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM3"
board = BoardShim(BoardIds.CYTON_BOARD, params)
```

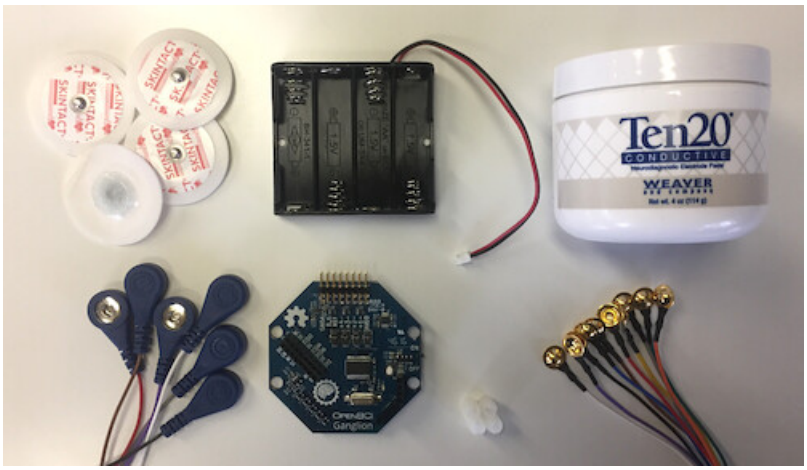
Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Devices like Raspberry Pi

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

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

### 1.2.2 Ganglion




Ganglion Getting Started Guide from OpenBCI

**To use Ganglion board you need a** dongle

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

**Also, for Macbooks without USB ports you may need to use specific USBC-USB dongles, some of them may lead to slow data streaming.**

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

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.GANGLION_BOARD`
- `serial_port`, e.g. COM4, /dev/ttyACM0, etc

- *optoinal:* `mac_address`, if not provided BrainFlow will try to autodiscover the device
- *optional:* `timeout`, timeout for device discovery, default is 15sec

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM4"
board = BoardShim(BoardIds.GANGLION_BOARD, params)
```

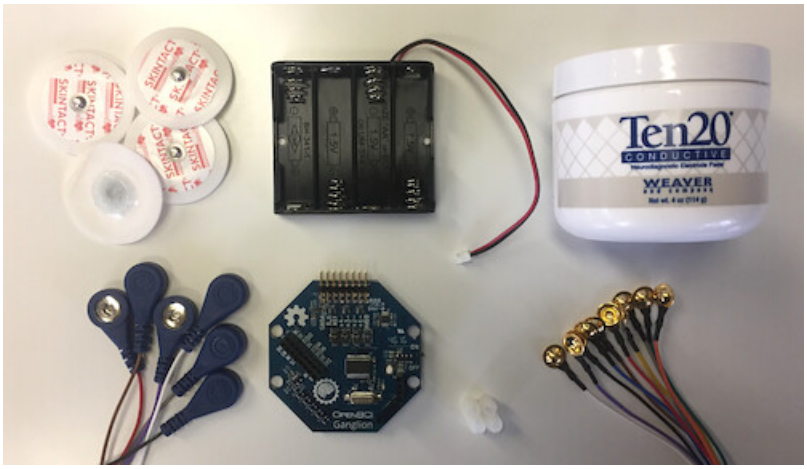
To get Ganglion's MAC address you can use:

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

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Devices like Raspberry Pi

### 1.2.3 Ganglion Native




Ganglion Getting Started Guide from OpenBCI

Unlike Ganglion board this BrainFlow board does not use BLED112 dongle, so you need to have BLE support on your device in order to use it.

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.GANGLION_NATIVE_BOARD`
- *optoinal:* `mac_address`, if not provided BrainFlow will try to autodiscover the device
- *optoinal:* `serial_number`, if not provided BrainFlow will try to autodiscover the device

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.GANGLION_NATIVE_BOARD, params)
```

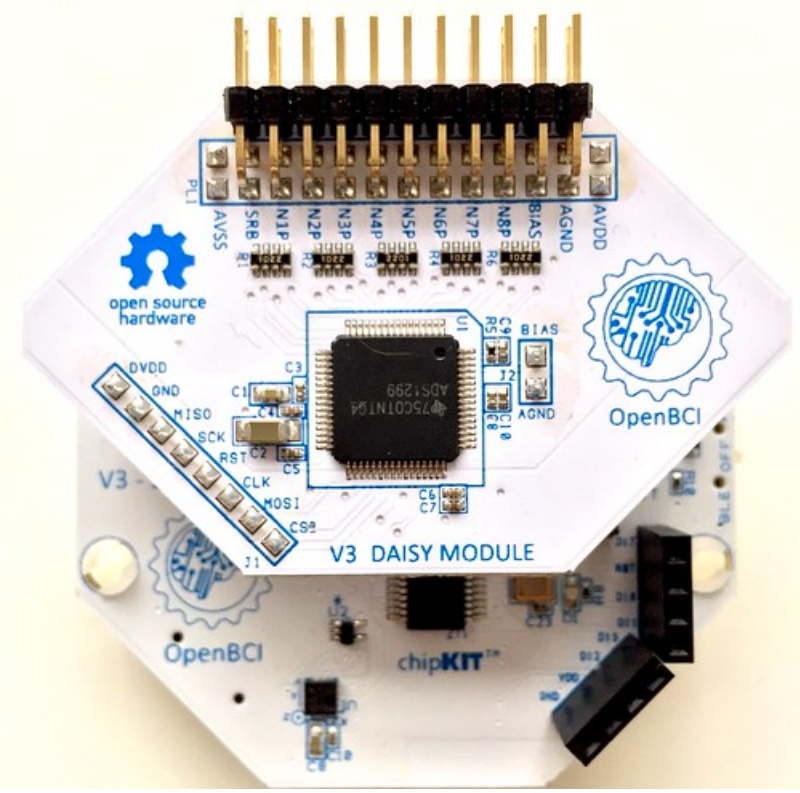
To get Ganglion's MAC address or device name you can use:

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

Supported platforms:

- Windows 10.0.19041.0+
- MacOS 10.15+, 12.0 to 12.2 have known issues while scanning, you need to update to 12.3+. On MacOS 12+ you may need to configure Bluetooth permissions for your appication
- Linux, compilation from source code probably will be needed
- Devices like Raspberry Pi

### 1.2.4 Cyton Daisy




CytonDaisy Getting Started Guide from OpenBCI

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.CYTON_DAISY_BOARD`
- `serial_port`, e.g. COM3, /dev/ttyUSB0, etc

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM3"
board = BoardShim(BoardIds.CYTON_DAISY_BOARD, params)
```

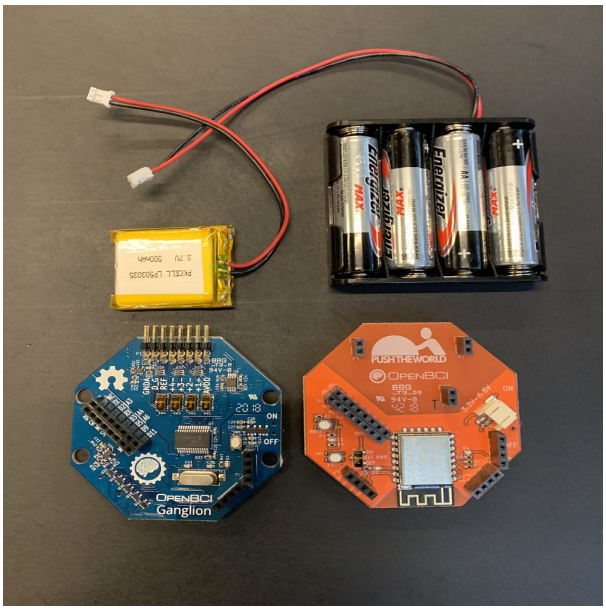
Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Devices like Raspberry Pi

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

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

### 1.2.5 Ganglion with WIFI Shield




WIFI Shield Getting Started Guide from OpenBCI

WIFI Shield Programming Guide from OpenBCI

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.GANGLION_WIFI_BOARD`
- `ip_port`, any local port which is currently free, e.g. 6789
- *optional:* `ip_address`, ip address of WIFI Shield, in direct mode it's 192.168.4.1. If not provided BrainFlow will try to use SSDP for discovery
- *optional:* `timeout`, timeout for device discovery, default is 10sec

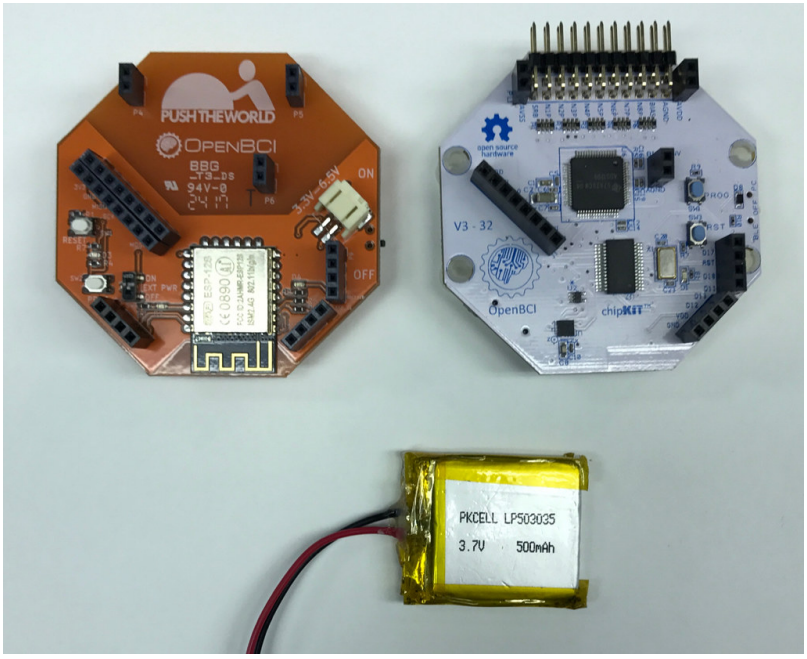
Initialization Example:

```
params = BrainFlowInputParams()
params.ip_port = 6987
params.ip_address = "192.168.4.1"
board = BoardShim(BoardIds.GANGLION_WIFI_BOARD, params)
```

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Android

### 1.2.6 Cyton with WIFI Shield




WIFI shield Getting Started Guide from OpenBCI

WIFI shield Programming Guide from OpenBCI

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.CYTON_WIFI_BOARD`
- `ip_port`, any local port which is currently free, e.g. 6789
- *optional:* `ip_address`, ip address of WIFI Shield, in direct mode it's 192.168.4.1. If not provided BrainFlow will try to use SSDP for discovery
- *optional:* `timeout`, timeout for device discovery, default is 10sec

Initialization Example:

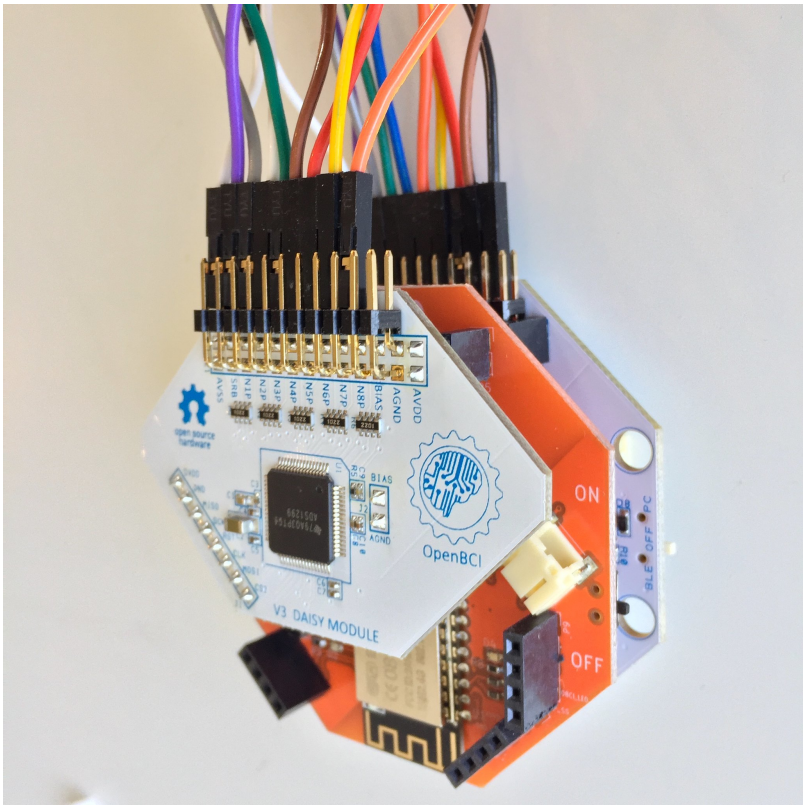
```
params = BrainFlowInputParams()
params.ip_port = 6987
params.ip_address = "192.168.4.1"
board = BoardShim(BoardIds.CYTON_WIFI_BOARD, params)
```

Supported platforms:

- Windows >= 8.1
- Linux

- MacOS
- Devices like Raspberry Pi
- Android

### 1.2.7 CytonDaisy with WIFI Shield




WIFI Shield Getting Started Guide from OpenBCI

WIFI Shield Programming Guide from OpenBCI

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.CYTON_DAISY_WIFI_BOARD`
- `ip_port`, any local port which is currently free, e.g. 6789
- *optional:* `ip_address`, ip address of WIFI Shield, in direct mode it's 192.168.4.1. If not provided BrainFlow will try to use SSDP for discovery
- *optional:* `timeout`, timeout for device discovery, default is 10sec

Initialization Example:

```
params = BrainFlowInputParams()
params.ip_port = 6987
params.ip_address = "192.168.4.1"
board = BoardShim(BoardIds.CYTON_DAISY_WIFI_BOARD, params)
```

Supported platforms:

- Windows >= 8.1
- Linux
- MacOS
- Devices like Raspberry Pi
- Android

## 1.3 NeuroMD

### 1.3.1 BrainBit


BrainBit website

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.BRAINBIT_BOARD`
- *optional:* `serial_number`, serial number of device, usually it's printed on the headset. Important if you have multiple devices in the same place
- *optional:* `timeout`, timeout for device discovery, default is 15sec

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.BRAINBIT_BOARD, params)
```

Supported platforms:

- Windows >= 10
- MacOS

Available commands for `config_board`:

- CommandStartSignal
- CommandStopSignal
- CommandStartResist
- CommandStopResist

### 1.3.2 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 create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.BRAINBIT_BLED_BOARD`
- `serial port`, e.g. COM4, /dev/ttyACM0
- *optional:* :code:mac_address`, mac address of BrainBit device, important if you have multiple devices in the same place

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM4"
board = BoardShim(BoardIds.BRAINBIT_BLED_BOARD, params)
```

To get BrainBit's MAC address you can use:

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

Supported platforms:

- Windows
- MacOS
- Linux
- Devices like Raspberry Pi

### 1.3.3 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:

- `BoardIds.CALLIBRI_EEG_BOARD`
- `BoardIds.CALLIBRI_EMG_BOARD`
- `BoardIds.CALLIBRI_ECG_BOARD`

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.CALLIBRI_EEG_BOARD`
- *optional:* `other_info`, to use electrodes connected vis USB write "ExternalSwitchInputMioUSB" to this field
- *optional:* `timeout`, timeout for device discovery, default is 15sec

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.CALLIBRI_EEG_BOARD, params)
```

Supported platforms:

- Windows >= 10
- MacOS

## 1.4 G.TEC

### 1.4.1 Unicorn

Unicorn website

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.UNICORN_BOARD`
- *optional:* `serial_number`, important if you have multiple devices in the same place

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.UNICORN_BOARD, params)
```

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

## 1.5 Neurosity

### 1.5.1 Notion 1

Neurosity website

Link to Neurosity Tutorial

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.NOTION_1_BOARD`
- *optional:* `serial_number` important if you have multiple devices in the same place

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.NOTION_1_BOARD, params)
```

Supported platforms:

- Windows
- Linux
- MacOS
- Devices like Raspberry Pi

*Note: On Windows you may need to disable firewall to allow broadcast messages. And since the device uses broadcasting it may not work in university network.*

### 1.5.2 Notion 2

Neurosity website

Link to Neurosity Tutorial

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.NOTION_2_BOARD`
- *optional:* `serial_number` important if you have multiple devices in the same place

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.NOTION_2_BOARD, params)
```

Supported platforms:

- Windows
- Linux
- MacOS
- Devices like Raspberry Pi

*Note: On Windows you may need to disable firewall to allow broadcast messages. And since the device uses broadcasting it may not work in university network.*

### 1.5.3 Crown


Neurosity website

Link to Neurosity Tutorial

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.CROWN_BOARD`
- *optional:* `serial_number` important if you have multiple devices in the same place

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.CROWN_BOARD, params)
```

Supported platforms:

- Windows
- Linux
- MacOS
- Devices like Raspberry Pi

*Note: On Windows you may need to disable firewall to allow broadcast messages. And since the device uses broadcasting it may not work in university network.*

## 1.6 OYMotion

### 1.6.1 gForcePro ArmBand


OYMotion website

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.GFORCE_PRO_BOARD`

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.GFORCE_PRO_BOARD, params)
```

Supported platforms:

- Windows

*Note: Unlike other boards it returns ADC values instead uV.*

### 1.6.2 gForceDual ArmBand

OYMotion website

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.GFORCE_DUAL_BOARD`

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.GFORCE_DUAL_BOARD, params)
```

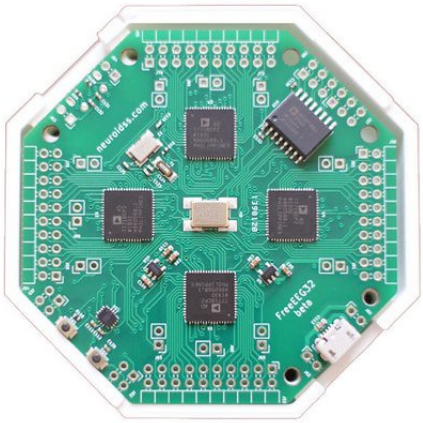
Supported platforms:

- Windows

*Note: Unlike other boards it returns ADC values instead uV.*

## 1.7 FreeEEG

### 1.7.1 FreeEEG32



CrowdSupply

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.FREEEEG32_BOARD`
- `serial_port`, e.g. COM3

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM3"
board = BoardShim(BoardIds.FREEEEG32_BOARD, params)
```

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

Supported platforms:

- Windows

- Linux
- MacOS
- Devices like Raspberry Pi

### 1.7.2 FreeEEG128

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.FREEEEG128_BOARD`
- `serial_port`, e.g. COM6

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM6"
board = BoardShim(BoardIds.FREEEEG128_BOARD, params)
```

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

Supported platforms:

- Windows
- Linux
- MacOS
- Devices like Raspberry Pi

## 1.8 Muse

### 1.8.1 Muse S BLED


Muse Website

To use this board you need to get BLED112 dongle.

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

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.MUSE_S_BLED_BOARD`
- `serial_port`, e.g. COM3, /dev/ttyACM0
- *optional:* `serial_number`, device name, can be printed on the Muse device or discoovered via mobile apps

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM3"
board = BoardShim(BoardIds.MUSE_S_BLED_BOARD, params)
```

Supported platforms:

- Windows
- MacOS
- Linux
- Devices like Raspberry Pi

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data, to enable 5th EEG channel use `board.config_board("p50")`
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data, enabled by default
- `BrainFlowPresets.ANCILLARY_PRESET`, it contains PPG data, to enable it use `board.config_board("p61")`

### 1.8.2 Muse 2 BLED


Muse Website

To use this board you need to get BLED112 dongle.

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

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.MUSE_2_BLED_BOARD`
- `serial_port`, e.g. COM3, /dev/ttyACM0
- *optional:* `serial_number`, device name, can be printed on the Muse device or discoovered via mobile apps

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM3"
board = BoardShim(BoardIds.MUSE_2_BLED_BOARD, params)
```

Supported platforms:

- Windows
- MacOS

- Linux
- Devices like Raspberry Pi

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data, to enable 5th EEG channel use `board.config_board("p50")`
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data, enabled by default
- `BrainFlowPresets.ANCILLARY_PRESET`, it contains PPG data, to enable it use `board.config_board("p50")`. It also enables 5th channel for EEG

### 1.8.3 Muse 2016 BLED


Muse Website

To use this board you need to get BLED112 dongle.

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

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.MUSE_2016_BLED_BOARD`
- `serial_port`, e.g. COM3, /dev/ttyACM0
- *optional:* `serial_number`, device name, can be printed on the Muse device or discoovered via mobile apps

Initialization Example:

```
params = BrainFlowInputParams()
params.serial_port = "COM3"
board = BoardShim(BoardIds.MUSE_2016_BLED_BOARD, params)
```

Supported platforms:

- Windows
- MacOS
- Linux
- Devices like Raspberry Pi

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data, enabled by default

### 1.8.4 Muse S


Muse Website

On Linux systems you may need to install *libdbus* and we recommend to compile BrainFlow from the source code:

```
sudo apt-get install libdbus-1-dev # for ubuntu
sudo yum install dbus-devel # for centos
python3 tools/build.py --ble # to compile
```

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.MUSE_S_BOARD`
- *optional:* `mac_address`, mac address of the device to connect
- *optional:* `serial_number`, device name, can be printed on the Muse device or discoovered via mobile apps

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.MUSE_S_BOARD, params)
```

Supported platforms:

- Windows 10.0.19041.0+
- MacOS 10.15+, 12.0 to 12.2 have known issues while scanning, you need to update to 12.3+. On MacOS 12+ you may need to configure Bluetooth permissions for your appication
- Linux, compilation from source code probably will be needed
- Devices like Raspberry Pi

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data, to enable 5th EEG channel use `board.config_board("p50")`
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data, enabled by default
- `BrainFlowPresets.ANCILLARY_PRESET`, it contains PPG data, to enable it use `board.config_board("p61")`

### 1.8.5 Muse 2


Muse Website

On Linux systems you may need to install *libdbus* and we recommend to compile BrainFlow from the source code:

```
sudo apt-get install libdbus-1-dev # for ubuntu
sudo yum install dbus-devel # for centos
python3 tools/build.py --ble # to compile
```

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.MUSE_2_BOARD`
- *optional:* `mac_address`, mac address of the device to connect
- *optional:* `serial_number`, device name, can be printed on the Muse device or discoovered via mobile apps

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.MUSE_2_BOARD, params)
```

Supported platforms:

- Windows 10.0.19041.0+
- MacOS 10.15+, 12.0 to 12.2 have known issues while scanning, you need to update to 12.3+. On MacOS 12+ you may need to configure Bluetooth permissions for your appication
- Linux, compilation from source code probably will be needed
- Devices like Raspberry Pi

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data, to enable 5th EEG channel use `board.config_board("p50")`
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data, enabled by default
- `BrainFlowPresets.ANCILLARY_PRESET`, it contains PPG data, to enable it use `board.config_board("p50")`. It also enables 5th channel for EEG

### 1.8.6 Muse 2016

### Muse Website

On Linux systems you may need to install *libdbus* and we recommend to compile BrainFlow from the source code:

```
sudo apt-get install libdbus-1-dev # for ubuntu
sudo yum install dbus-devel # for centos
python3 tools/build.py --ble # to compile
```

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.MUSE_2016_BOARD`
- *optional:* `mac_address`, mac address of the device to connect
- *optional:* `serial_number`, device name, can be printed on the Muse device or discoovered via mobile apps

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.MUSE_2016_BOARD, params)
```

Supported platforms:

- Windows 10.0.19041.0+
- MacOS 10.15+, 12.0 to 12.2 have known issues while scanning, you need to update to 12.3+. On MacOS 12+ you may need to configure Bluetooth permissions for your appication
- Linux, compilation from source code probably will be needed
- Devices like Raspberry Pi

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data, enabled by default

## 1.9 Ant Neuro


Ant Website

Ant Neuro has many devices and all of them are supported by BrainFlow:

- ANT_NEURO_EE_410_BOARD
- ANT_NEURO_EE_411_BOARD
- ANT_NEURO_EE_430_BOARD
- ANT_NEURO_EE_211_BOARD
- ANT_NEURO_EE_212_BOARD
- ANT_NEURO_EE_213_BOARD
- ANT_NEURO_EE_214_BOARD
- ANT_NEURO_EE_215_BOARD
- ANT_NEURO_EE_221_BOARD
- ANT_NEURO_EE_222_BOARD
- ANT_NEURO_EE_223_BOARD

- ANT_NEURO_EE_224_BOARD
- ANT_NEURO_EE_225_BOARD
- ANT_NEURO_EE_511_BOARD

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.ANT_NEURO_EE_410_BOARD, params)
```

Supported platforms:

- Windows
- Linux

Available commands:

- Set sampling rate: `board.config_board("sampling_rate:500")`, for available values check docs from Ant Neuro.

For more information about Ant Neuro boards please refer to their User Manual.

## 1.10 Enophone

### 1.10.1 Enophone Headphones


Enophone website

**You need to pair your device first.**

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.ENOPHONE_BOARD`
- `mac_address`, it's optional for some OSes. Windows and MacOS can autodiscover paired devices, Linux cannot

Initialization Example:

```
params = BrainFlowInputParams()
params.mac_address = "F4:0E:11:75:76:78"
board = BoardShim(BoardIds.ENOPHONE_BOARD, params)
```

Supported platforms:

- Windows

- Linux
- MacOS
- Devices like Raspberry Pi

Steps to find MAC address:

- On Windows: open device manager, navigate to enophone device, click properties, details, and select Bluetooth Address
- On Linux: install bluez-utils and run `bluetoothctl paired-devices`
- On MacOS: run `system_profiler SPBluetoothDataType`

**On Linux in order to compile and use it you may need to install :code:`libbluetooth-dev` for Debian like systems from :code:`apt-get` and :code:`bluez-libs-devel` for Fedora like systems from :code:`dnf`.**

## 1.11 BrainAlive

### 1.11.1 BrainAlive Device


BrainAlive Website

On Linux systems you may need to install *libdbus* and we recommend to compile BrainFlow from the source code:

```
sudo apt-get install libdbus-1-dev # for ubuntu
sudo yum install dbus-devel # for centos
python3 tools/build.py --ble # to compile
```

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.BRAINALIVE_BOARD`
- *optional:* `mac_address`, mac address of the device to connect
- *optional:* `serial_number`, device name

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.BRAINALIVE_BOARD, params)
```

Supported platforms:

- Windows 10.0.19041.0+
- MacOS 10.15+
- Linux, compilation from source code probably will be needed
- Devices like Raspberry Pi

## 1.12 Mentalab

### 1.12.1 Explore 4 Channels Board

Mentalab website

**You need to pair your device first.**

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.EXPLORE_4_CHAN_BOARD`
- `mac_address`, it's optional for some OSes. Windows and MacOS can autodiscover paired devices, Linux cannot

Initialization Example:

```
params = BrainFlowInputParams()
params.mac_address = "F4:0E:11:75:76:78"
board = BoardShim(BoardIds.EXPLORE_4_CHAN_BOARD, params)
```

Supported platforms:

- Windows
- Linux
- MacOS
- Devices like Raspberry Pi

**On Linux in order to compile and use it you may need to install :code:`libbluetooth-dev` for Debian like systems from :code:`apt-get` and :code:`bluez-libs-devel` for Fedora like systems from :code:`dnf`.**

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data
- `BrainFlowPresets.ANCILLARY_PRESET`, it contains battery and temperature data

Steps to find MAC address:

- On Windows: open device manager, navigate to explore device, click properties, details, and select Bluetooth Address
- On Linux: install bluez-utils and run `bluetoothctl paired-devices`
- On MacOS: run `system_profiler SPBluetoothDataType`

Steps to connect:

- Enable device and Pair it with your laptop using bluetooth settings
- Ensure that blue LED is blinking before calling `board.prepare_session()`
- If you see green LED probably you need to reboot a devce

### 1.12.2 Explore 8 Channels Board

Mentalab website

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

**You need to pair your enophone device first.**

- `BoardIds.EXPLORE_8_CHAN_BOARD`
- `mac_address`, it's optional for some OSes. Windows and MacOS can autodiscover paired Enophone devices, Linux cannot

Initialization Example:

```
params = BrainFlowInputParams()
params.mac_address = "F4:0E:11:75:76:78"
board = BoardShim(BoardIds.EXPLORE_8_CHAN_BOARD, params)
```

Supported platforms:

- Windows
- Linux
- MacOS
- Devices like Raspberry Pi

**On Linux in order to compile and use it you may need to install :code:`libbluetooth-dev` for Debian like systems from :code:`apt-get` and :code:`bluez-libs-devel` for Fedora like systems from :code:`dnf`.**

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains EEG data
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains Gyro and Accel data

- `BrainFlowPresets.ANCILLARY_PRESET`, it contains battery and temperature data

Steps to find MAC address:

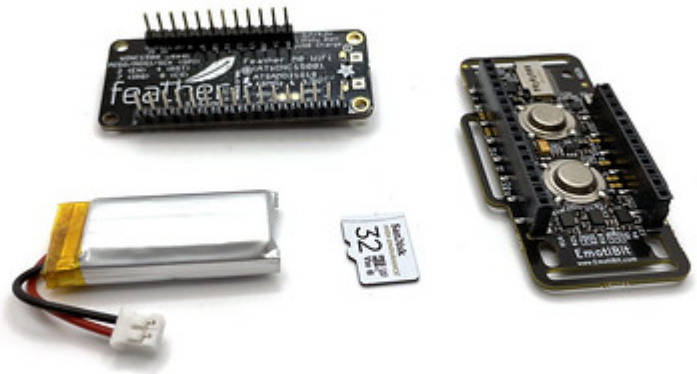
- On Windows: open device manager, navigate to explore device, click properties, details, and select Bluetooth Address
- On Linux: install bluez-utils and run `bluetoothctl paired-devices`
- On MacOS: run `system_profiler SPBluetoothDataType`

Steps to connect:

- Enable device and Pair it with your laptop using bluetooth settings
- Ensure that blue LED is blinking before calling `board.prepare_session()`
- If you see green LED probably you need to reboot a devce

## 1.13 EmotiBit

### 1.13.1 EmotiBit board



EmotiBit Website

To create such board you need to specify the following board ID and fields of BrainFlowInputParams object:

- `BoardIds.EMOTIBIT_BOARD`
- *optional:* `ip_address`, you can provide *broadcast* ip address of the network with EmotiBit device, e.g. 192.168.178.255. If not provided BrainFlow will try to autodiscover the network and it may take a little longer.

Initialization Example:

```
params = BrainFlowInputParams()
board = BoardShim(BoardIds.EMOTIBIT_BOARD, params)
```

Supported platforms:

- Windows

- MacOS
- Linux
- Devices like Raspberry Pi

Available *BrainFlow Presets*:

- `BrainFlowPresets.DEFAULT_PRESET`, it contains accelerometer, gyroscope and magnetometer data
- `BrainFlowPresets.AUXILIARY_PRESET`, it contains PPG data
- `BrainFlowPresets.ANCILLARY_PRESET`, it contains EDA and temperature data

CHAPTER

TWO

# INSTALLATION INSTRUCTIONS

## 2.1 Precompiled libraries in package managers(Nuget, PYPI, etc)

Core part of BrainFlow is written in C/C++ and distributed as dynamic libraries, for some programming languages we publish packages with precompiled libraries to package managers like Nuget or PYPI.

C/C++ code should be compiled for each CPU architecture and for each OS and we cannot cover all possible cases, as of right now we support:

- x64 libraries for Windows starting from 8.1, for some devices newer version of Windows can be required
- x64 libraries for Linux, they are compiled inside manylinux docker container
- x64/ARM libraries for MacOS, they are universal binaries

If your CPU and OS is not listed above(e.g. Raspberry Pi or Windows with ARM) you still can use BrainFlow, but you need to compile it by youself first. See *Compilation of Core Module and C++ Binding* for details.

## 2.2 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 first compile the core module (*Compilation of Core Module and C++ Binding*). Then run

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

## 2.3 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
- 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 Mono and other dependcies on your system
- Build it using dotnet command
- **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
python tools/build.py
# install dependencies
sudo dnf install nuget
sudo dnf install mono-devel
sudo dnf install mono-complete
sudo dnf install monodevelop
sudo dnf install dotnet-sdk-6.0
sudo dnf install dotnet-runtime-6.0
sudo dnf install dotnet-sdk-3.1
sudo dnf install dotnet-runtime-3.1
# build solution
dotnet build csharp_package/brainflow/brainflow.sln
# run tests
export LD_LIBRARY_PATH=/home/andreyparfenov/brainflow/installed/lib/
mono csharp_package/brainflow/examples/denoising/bin/Debug/denoising.exe
```

## 2.4 R

R binding is based on reticulate package and calls Python , 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.

## 2.5 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 (*Compilation of Core Module and C++ Binding*) 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.

## 2.6 Matlab

Steps to setup Matlab binding for BrainFlow:

- Compile Core Module, using the instructions in *Compilation of Core Module and C++ Binding*. 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
- If you see errors you may need to configure Matlab to use C++ compiler instead C, install Visual Studio 2017 or newer(for Windows) and run this command in Matlab terminal `mex -setup cpp`, you need to select Visual Studio Compiler from the list. More info can be found here.

## 2.7 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.

## 2.8 Typescript

You can install BrainFlow using next command without compilation

```
npm install brainflow
```

If you want to install it from source files or build unreleased version from Github, you should first compile the core module (*Compilation of Core Module and C++ Binding*). Then run

```
cd nodejs_package
npm install
```

## 2.9 Rust

You can build Rust binding locally using commands below, but you need to compile C/C++ code first

```
cd rust_package
cd brainflow
cargo build --features generate_binding
```

## 2.10 Docker Image

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

All bindings except Matlab are preinstalled there.

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

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

If your devices uses TCP/IP 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:latest
# run docker container with serial port /dev/ttyUSB0
docker run -it --device /dev/ttyUSB0 brainflow/brainflow:latest /bin/bash
# run docker container for boards which use networking
docker run -it --network host brainflow/brainflow:latest /bin/bash
```

## 2.11 Compilation of Core Module and C++ Binding

### 2.11.1 Windows

- Install CMake>=3.16 you can install it from PYPI via pip or from CMake website
- Install Visual Studio 2019(preferred) or Visual Studio 2017. Other versions may work but not tested
- In VS installer make sure you selected "Visual C++ ATL support"
- Build it as a standard CMake project, you don't need to set any options

If you are not familiar with CMake you can use build.py :

```
# install python3 and run
python -m pip install cmake
cd tools
python build.py
# to get info about args and configure your build you can run
python build.py --help
```

### 2.11.2 Linux

- Install CMake>=3.16 you can install it from PYPI via pip, via package managers for your OS(apt, dnf, etc) or from CMake website
- If you are going to distribute compiled Linux libraries you HAVE to build it inside manylinux Docker container
- Build it as a standard CMake project, you don't need to set any options
- You can use any compiler but for Linux we test only GCC

If you are not familiar with CMake you can use build.py :

```
python3 -m pip install cmake
cd tools
python3 build.py
# to get info about args and configure your build you can run
python3 build.py --help
```

### 2.11.3 MacOS

- Install CMake>=3.16 you can install it from PYPI via pip, using `brew` or from CMake website
- Build it as a standard CMake project, you don't need to set any options
- You can use any compiler but for MacOS we test only Clang

If you are not familiar with CMake you can use build.py :

```
python3 -m pip install cmake
cd tools
python3 build.py
# to get info about args and configure your build you can run
python3 build.py --help
```

## 2.12 Android

To check supported boards for Android visit *Supported Boards*

### 2.12.1 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*

Prebuild libraries for *jniLibs.zip* are complied using:

- Android NDK 25.1.8937393
- *-DANDROID_NATIVE_API_LEVEL=android-24*

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>
```

### 2.12.2 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=D:\workspace\android-ndk-r25b\build\cmake\android.
↪toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-24 -DANDROID_ABI=arm64-v8a ..
# for armeabi-v7a
cmake -G Ninja -DCMAKE_TOOLCHAIN_FILE=D:\workspace\android-ndk-r25b\build\cmake\android.
↪toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-24 -DANDROID_ABI=armeabi-v7a ..
```

(continues on next page)

(continued from previous page)

```
# for x86_64
cmake -G Ninja -DCMAKE_TOOLCHAIN_FILE=D:\workspace\android-ndk-r25b\build\cmake\android.
↪toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-24 -DANDROID_ABI=x86_64 ..
# for x86
cmake -G Ninja -DCMAKE_TOOLCHAIN_FILE=D:\workspace\android-ndk-r25b\build\cmake\android.
↪toolchain.cmake -DANDROID_NATIVE_API_LEVEL=android-24 -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
NOTION_1_BOARD = 13
NOTION_2_BOARD = 14
GFORCE_PRO_BOARD = 16
FREEEEG32_BOARD = 17
BRAINBIT_BLED_BOARD = 18
GFORCE_DUAL_BOARD = 19
GALEA_SERIAL_BOARD = 20
MUSE_S_BLED_BOARD = 21
MUSE_2_BLED_BOARD = 22
CROWN_BOARD = 23
ANT_NEURO_EE_410_BOARD = 24
ANT_NEURO_EE_411_BOARD = 25
ANT_NEURO_EE_430_BOARD = 26
ANT_NEURO_EE_211_BOARD = 27
ANT_NEURO_EE_212_BOARD = 28
ANT_NEURO_EE_213_BOARD = 29
ANT_NEURO_EE_214_BOARD = 30
ANT_NEURO_EE_215_BOARD = 31
ANT_NEURO_EE_221_BOARD = 32
ANT_NEURO_EE_222_BOARD = 33
ANT_NEURO_EE_223_BOARD = 34
ANT_NEURO_EE_224_BOARD = 35
ANT_NEURO_EE_225_BOARD = 36
ENOPHONE_BOARD = 37
MUSE_2_BOARD = 38
MUSE_S_BOARD = 39
BRAINALIVE_BOARD = 40
MUSE_2016_BOARD = 41
MUSE_2016_BLED_BOARD = 42
EXPLORE_4_CHAN_BOARD = 44
EXPLORE_8_CHAN_BOARD = 45
```

GANGLION_NATIVE_BOARD = 46

EMOTIBIT_BOARD = 47

GALEA_BOARD_V4 = 48

GALEA_SERIAL_BOARD_V4 = 49

NTL_WIFI_BOARD = 50

ANT_NEURO_EE_511_BOARD = 51

FREEEEG128_BOARD = 52

AAVAA_V3_BOARD = 53

EXPLORE_PLUS_8_CHAN_BOARD = 54

EXPLORE_PLUS_32_CHAN_BOARD = 55

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

Bases: enum.IntEnum

Enum to store Ip Protocol types

NO_IP_PROTOCOL = 0

UDP = 1

TCP = 2

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

Bases: enum.IntEnum

Enum to store presets

DEFAULT_PRESET = 0

AUXILIARY_PRESET = 1

ANCILLARY_PRESET = 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_address_aux** (*str*) – ip address is used for boards which reads data from socket connection
- **ip_address_anc** (*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_port_aux** (*int*) – ip port for socket connection, for some boards where we know it in front you dont need this parameter
- **ip_port_anc** (*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 IpProtocolTypes enum
- **other_info** (*str*) – other info
- **serial_number** (*str*) – serial number
- **file** (*str*) – file
- **file_aux** (*str*) – file
- **file_anc** (*str*) – file

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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → int
get sampling rate for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → int

get package num channel for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → int

get battery channel for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → int

get number of rows in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → int

get timestamp channel in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → int

get marker channel in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[str]

get names of EEG channels in 10-20 system if their location is fixed

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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_presets**(*board_id: int*) → List[str]

get available presets for this board id

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

**Returns** presets for this board id

**Return type** List[str]

:raises BrainFlowError

**classmethod get_version**() → str

get version of brainflow libraries

**Returns** version

**Return type** str

:raises BrainFlowError

**classmethod get_board_descr**(*board_id: int*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*)

get board description as json

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → str

get device name

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of eeg channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of exg channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of emg channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of ecg channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of eog channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of eda channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of ppg channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of accel channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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_rotation_channels**(*board_id: int*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of rotation channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**Returns** list of rotation 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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of analog channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of gyro channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of other channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of temperature channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of resistance channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**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

**classmethod get_magnetometer_channels**(*board_id: int*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → List[int]

get list of magnetometer channels in resulting data table for a board

**Parameters**

- **board_id** (*int*) – Board Id
- **preset** (*int*) – preset

**Returns** list of magnetometer 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 release_all_sessions**() → None
release all prepared sessions

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

**add_streamer**(*streamer_params: str*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → None
Add streamer

**Parameters**

- **preset** (*int*) – preset
- **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”

**delete_streamer**(*streamer_params: str*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → None
Delete streamer

**Parameters**

- **preset** (*int*) – preset
- **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”

**start_stream**(*num_samples: int = 450000*, *streamer_params: Optional[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**(*num_samples: int*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]
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
- **preset** (*int*) – preset

**Returns** latest data from a board

**Return type** NDArray[Float64]

**get_board_data_count**(*preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → int
Get num of elements in ringbuffer

**Parameters** **preset** (*int*) – preset

**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*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → None
Insert Marker to Data Stream

**Parameters**

- **value** (*float*) – value to insert
- **preset** (*int*) – preset

**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**(*num_samples=None*, *preset: int = <BrainFlowPresets.DEFAULT_PRESET: 0>*) → nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]
Get board data and remove data from ringbuffer

**Parameters**

- **num_samples** (*int*) – number of packages to get
- **preset** (*int*) – preset

**Returns** all data from a board if num_samples is None, num_samples packages or less if not None

**Return type** NDArray[Float64]

**config_board**(*config*) → str
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

**config_board_with_bytes**(*bytes_to_send*) → None
Use this method carefully and only if you understand what you are doing

**Parameters** **bytes_to_send** – bytes to send

### 3.1.2 brainflow.exit_codes

**exception** brainflow.exit_codes.**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.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**

**BUTTERWORTH_ZERO_PHASE = 3**

**CHEBYSHEV_TYPE_1_ZERO_PHASE = 4**

**BESSEL_ZERO_PHASE = 5**

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.**WindowOperations**(*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

**NO_DETREND = 0**

**CONSTANT = 1**

**LINEAR = 2**

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

Enum to store noise types

**FIFTY = 0**

**SIXTY = 1**

**FIFTY_AND_SIXTY = 2**

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

Enum to store all supported wavelet denoising methods

**VISUSHRINK = 0**

**SURESHRINK = 1**

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

Enum to store all supported thresholding types

**SOFT = 0**

**HARD = 1**

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

Enum to store all supported wavelet extension types

**SYMMETRIC = 0**

**PERIODIC = 1**

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

Enum to store all supported values for noise estemation levels in wavelet denoising

**FIRST_LEVEL = 0**

**ALL_LEVELS = 1**

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

Enum to store all supported wavelets

**HAAR = 0**

**DB1 = 1**

**DB2 = 2**

**DB3 = 3**

**DB4 = 4**

**DB5 = 5**

**DB6 = 6**

**DB7 = 7**

**DB8 = 8**

**DB9 = 9**

**DB10 = 10**

**DB11 = 11**

**DB12 = 12**

**DB13 = 13**

**DB14 = 14**

**DB15 = 15**

**BIOR1_1 = 16**

**BIOR1_3 = 17**

**BIOR1_5 = 18**

**BIOR2_2 = 19**

**BIOR2_4 = 20**

**BIOR2_6 = 21**

**BIOR2_8 = 22**

**BIOR3_1 = 23**

**BIOR3_3 = 24**

**BIOR3_5 = 25**

**BIOR3_7 = 26**

**BIOR3_9 = 27**

**BIOR4_4 = 28**

**BIOR5_5 = 29**

**BIOR6_8 = 30**

**COIF1 = 31**

**COIF2 = 32**

**COIF3 = 33**

**COIF4 = 34**

**COIF5 = 35**

**SYM2 = 36**

**SYM3 = 37**

**SYM4 = 38**

**SYM5 = 39**

**SYM6 = 40**

**SYM7 = 41**

**SYM8 = 42**

**SYM9 = 43**

**SYM10 = 44**

**class** brainflow.data_filter.**DataFilter**

Bases: object

DataFilter class contains methods for signal processig

**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_data_logger, enable_dev_data_logger or disable_data_logger

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

**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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *cutoff: float*, *order: int*, *filter_type: int*, *ripple: float*) → None
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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *cutoff: float*, *order: int*, *filter_type: int*, *ripple: float*) → None
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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *start_freq: float*, *stop_freq: float*, *order: int*, *filter_type: int*, *ripple: float*) → None
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
- **start_freq** (*float*) – start frequency
- **stop_freq** (*float*) – stop frequency
- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum

- **ripple** (*float*) – ripple value for Chebyshev filter

**classmethod perform_bandstop**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *start_freq: float*, *stop_freq: float*, *order: int*, *filter_type: int*, *ripple: float*) → None

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
- **start_freq** (*float*) – start frequency
- **stop_freq** (*float*) – stop frequency
- **order** (*int*) – filter order
- **filter_type** (*int*) – filter type from special enum
- **ripple** (*float*) – ripple value for Chebyshev filter

**classmethod remove_environmental_noise**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *noise_type: float*) → None

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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *period: int*, *operation: int*) → None

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 calc_stddev**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*)

calc stddev

**Parameters** **data** (*NDArray[Float64]*) – input array

**Returns** stddev

**Return type** float

**classmethod get_railed_percentage**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *gain: int*)

get railed percentage

**Parameters**

- **data** (*NDArray[Float64]*) – input array

- **gain** (*int*) – gain

**Returns** railed percentage

**Return type** float

**classmethod get_oxygen_level**(*ppg_ir: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *ppg_red: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *coef1=1.5958422*, *coef2=- 34.6596622*, *coef3=112.6898759*)

get oxygen level from ppg

**Parameters**

- **ppg_ir** (*NDArray[Float64]*) – input array
- **ppg_red** (*NDArray[Float64]*) – input array
- **sampling_rate** (*int*) – sampling rate

**Returns** oxygen level

**Return type** float

**classmethod get_heart_rate**(*ppg_ir: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *ppg_red: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *fft_size: int*)

get heart rate

**Parameters**

- **ppg_ir** (*NDArray[Float64]*) – input array
- **ppg_red** (*NDArray[Float64]*) – input array
- **sampling_rate** (*int*) – sampling rate
- **fft_size** (*int*) – recommended 8192

**Returns** heart rate

**Return type** float

**classmethod perform_downsampling**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *period: int*, *operation: int*) → nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]

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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float], wavelet: int, decomposition_level: int, extension_type=<WaveletExtensionTypes.SYMMETRIC: 0>*) → Tuple

perform wavelet transform

**Parameters**

- **data** (*NDArray[Float64]*) – initial data
- **wavelet** (*int*) – use WaveletTypes enum
- **decomposition_level** (*int*) – level of decomposition
- **extension_type** – extension type, use WaveletExtensionTypes

**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 restore_data_from_wavelet_detailed_coeffs**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float], wavelet, decomposition_level, level_to_restore*)

restore data from a single wavelet coeff

**Parameters**

- **data** (*NDArray[Float64]*) – initial data
- **wavelet** (*int*) – use WaveletTypes enum
- **decomposition_level** (*int*) – level of decomposition
- **level_to_restore** (*int*) – level of coeffs

**Returns**

**Return type** NDArray[Float64]

**classmethod detect_peaks_z_score**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float], lag=5, threshold=3.5, influence=0.1*)

z score algorithm for peak detection

**Parameters**

- **data** (*NDArray[Float64]*) – initial data
- **lag** (*int*) – window size for averaging
- **threshold** (*float*) – in stddev units
- **influence** (*float*) – contribution of peaks to mean value, between 0 and 1

**Returns**

**Return type** NDArray[Float64]

**classmethod perform_inverse_wavelet_transform**(*wavelet_output: Tuple*, *original_data_len: int*, *wavelet: int*, *decomposition_level: int*, *extension_type=<WaveletExtensionTypes.SYMMETRIC: 0>*) → nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]

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** (*int*) – use WaveletTypes enum
- **decomposition_level** (*int*) – level of decomposition
- **extension_type** – extension type, use WaveletExtensionTypes

**Returns** restored data

**Return type** NDArray[Float64]

**classmethod perform_wavelet_denoising**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float], wavelet: int, decomposition_level: int, wavelet_denoising=<WaveletDenoisingTypes.SURESHRINK: 1>, threshold=<ThresholdTypes.HARD: 1>, extension_type=<WaveletExtensionTypes.SYMMETRIC: 0>, noise_level=<NoiseEstimationLevelTypes.FIRST_LEVEL: 0>*) → None

perform wavelet denoising

**Parameters**

- **data** (*NDArray[Float64]*) – data to denoise
- **wavelet** (*int*) – use WaveletTypes enum
- **decomposition_level** (*int*) – decomposition level
- **wavelet_denoising** (*int*) – use WaveletDenoisingTypes enum
- **threshold** (*int*) – use ThresholdTypes enum
- **extension_type** (*int*) – use WaveletExtensionTypes enum
- **noise_level** (*int*) – use NoiseEstimationLevelTypes enum

**classmethod get_csp**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *labels: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*) → Tuple

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**(*window_function: int*, *window_len: int*) → nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]

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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *window: int*) → nptyping.types._ndarray.NDArray[None, nptyping.types._complex.Complex128]

perform direct fft

**Parameters**

- **data** (*NDArray[Float64]*) – data for fft, len of data must be even
- **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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *sampling_rate: int*, *window: int*) → Tuple

calculate PSD

**Parameters**

- **data** (*NDArray[Float64]*) – data to calc psd, len of data must be even
- **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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *nfft: int*, *overlap: int*, *sampling_rate: int*, *window: int*) → Tuple

calculate PSD using Welch method

**Parameters**

- **data** (*NDArray[Float64]*) – data to calc psd
- **nfft** (*int*) – FFT Window size, must be even
- **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**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]*, *detrend_operation: int*) → None

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** (*tuple*) – 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, bands are 1-4,4-8,8-13,13-30,30-50

**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 get_custom_band_powers**(*data: nptyping.types._ndarray.NDArray*, *bands: List*, *channels: List*, *sampling_rate: int*, *apply_filter: bool*) → Tuple

calculate avg and stddev of BandPowers across selected channels

**Parameters**

- **data** (*NDArray*) – 2d array for calculation
- **bands** (*List*) – List of typles with bands to use. E.g [(1.5, 4.0), (4.0, 8.0), (8.0, 13.0), (13.0, 30.0), (30.0, 45.0)]
- **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_ica**(*data: nptyping.types._ndarray.NDArray*, *num_components: int*, *channels=None*) → Tuple

perform ICA

**Parameters**

- **data** (*NDArray*) – 2d array for calculation
- **num_components** (*int*) – number of components
- **channels** (*List*) – channels - rows of data array which should be used for calculation, if None use all

**Returns** w, k, a, s matrixes as a tuple

**Return type** tuple

**classmethod perform_ifft**(*data: nptyping.types._ndarray.NDArray[None, nptyping.types._complex.Complex128]*) → nptyping.types._ndarray.NDArray[None, nptyping.types._number.Float]

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

**classmethod get_version**() → str

get version of brainflow libraries

**Returns** version

**Return type** str

:raises BrainFlowError

**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

### 3.1.4 brainflow.ml_model

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

Enum to store all supported metrics

**MINDFULNESS = 0**

**RESTFULNESS = 1**

**USER_DEFINED = 2**

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

Enum to store all supported classifiers

**DEFAULT_CLASSIFIER = 0**

**DYN_LIB_CLASSIFIER = 1**

**ONNX_CLASSIFIER = 2**

**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** (*str*) – additional information
- **output_name** (*str*) – output node name
- **max_array_size** (*int*) – max array size to preallocate

**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 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_ml_logger, enable_dev_ml_logger or disable_ml_logger

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

**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

**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 release_all**() → None
release all classifiers

**classmethod get_version**() → str
get version of brainflow libraries

**Returns** version

**Return type** str

:raises BrainFlowError

**prepare**() → None
prepare classifier

**release**() → None
release classifier

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

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

**Returns** metric value

**Return type** List

## 3.2 C++ API Reference

### 3.2.1 BoardShim class

class **BoardShim**
*BoardShim* class to communicate with a board.

### Public Functions

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

void **add_streamer**(std::string streamer_params, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
add streamer

**Parameters** **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"

void **delete_streamer**(std::string streamer_params, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
delete streamer

**Parameters** **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, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
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**(int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get number of packages in ringbuffer

BrainFlowArray<double, 2> **get_board_data**(int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get all collected data and flush it from internal buffer

BrainFlowArray<double, 2> **get_board_data**(int num_datapoints, int preset)
get required amount of datapoints or less and flush it from internal buffer

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

void **config_board_with_bytes**(const char *bytes, int len)
send raw bytes to a board, not implemented for majority of devices, not recommended to use

void **insert_marker**(double value, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
insert marker in data stream

### Public Static Functions

static void **disable_board_logger**()
disable BrainFlow loggers

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

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

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

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

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

static json **get_board_descr**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get board description as json

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If board id is not valid exit code is UNSUPPORTED_BOARD_ERROR

static int **get_sampling_rate**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get sampling rate for this board

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static int **get_package_num_channel**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row index which holds package nums

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static int **get_timestamp_channel**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row index which holds timestamps

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static int **get_marker_channel**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row index which holds markers

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static int **get_battery_channel**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row index which holds battery level info

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static int **get_num_rows**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get number of rows in returned from *get_board_data()* 2d array

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::string **get_device_name**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get device name

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<std::string> **get_eeg_names**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get eeg channel names in 10-20 system for devices with fixed electrode locations

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_eeg_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row indices which hold EEG data, for some board we can not split EEG\EMG...

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

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

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_ecg_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row indices which hold ECG data, for some board we can not split EEG\EMG...

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_eog_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row indices which hold EOG data, for some board we can not split EEG\EMG...

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_exg_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)

get row indices which hold EXG data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_ppg_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)

get row indices which hold PPG data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_eda_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)

get row indices which hold EDA data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_accel_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)

get row indices which hold accel data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_rotation_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)

get row indices which hold rotation data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_analog_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)

get row indices which hold rotation calib data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_gyro_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)

get row indices which hold gyro data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_other_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row indices which hold other information

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_temperature_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row indices which hold temperature data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_resistance_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row indices which hold resistance data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static std::vector<int> **get_magnetometer_channels**(int board_id, int preset = (int)BrainFlowPresets::DEFAULT_PRESET)
get row indices which hold magnetometer data

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

static void **release_all_sessions**()
release all currently prepared session

static std::string **get_version**()
get brainflow version

static std::vector<int> **get_board_presets**(int board_id)
get available presets for this board

**Parameters** **board_id** – board id of your device

**Throws** **BrainFlowException** –

### 3.2.2 DataFilter class

class **DataFilter**
*DataFilter* class to perform signal processing.

### Public Static Functions

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

static void **disable_data_logger**()
disable Data loggers

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

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

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

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

static 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

static 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

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

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

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

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

static 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

static std::pair<double*, int*> **perform_wavelet_transform**(double *data, int data_len, int wavelet, int decomposition_level, int extension_type = (int)WaveletExtensionTypes::SYMMETRIC)
perform wavelet transform

**Parameters**

- **data** – input array, any size
- **data_len** – length of input array
- **wavelet** – use WaveletTypes enum
- **decomposition_level** – level of decomposition in wavelet transform
- **extension** – use WaveletExtensionTypes enum

**Returns** 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

static double ***perform_inverse_wavelet_transform**(std::pair<double*, int*> wavelet_output, int original_data_len, int wavelet, int decomposition_level, int extension_type = (int)WaveletExtensionTypes::SYMMETRIC)

performs inverse wavelet transform

static void **perform_wavelet_denoising**(double *data, int data_len, int wavelet, int decomposition_level, int wavelet_denoising = (int)WaveletDenoisingTypes::SURESHRINK, int threshold = (int)ThresholdTypes::HARD, int extenstion_type = (int)WaveletExtensionTypes::SYMMETRIC, int noise_level = (int)NoiseEstimationLevelTypes::FIRST_LEVEL)

perform wavelet denoising

**Parameters**

- **data** – input array, any size
- **data_len** – length of input array
- **wavelet** – use WaveletTypes enum
- **decomposition_level** – level of decomposition in wavelet transform
- **wavelet_denoising** – use WaveletDenoisingTypes enum
- **threshold** – use ThresholdTypes enum
- **extension** – use WaveletExtensionTypes enum
- **noise_level** – use NoiseEstimationLevelTypes enum

static void **restore_data_from_wavelet_detailed_coeffs**(double *data, int data_len, int wavelet, int decomposition_level, int level_to_restore, double *output)

restore data from selected detailed coeffs

static void **detect_peaks_z_score**(double *data, int data_len, int lag, double threshold, double influence, double *output)

z score peak detection, more info https://stackoverflow.com/a/22640362

static 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

**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

**Returns** 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

static double *get_window(int window_function, int window_len)
perform data windowing

static std::complex<double> *perform_fft(double *data, int data_len, int window, int *fft_len)
perform direct fft

**Parameters**

- **data** – input array
- **data_len** – must be even
- **window** – window function
- **fft_len** – output fft len(data_len / 2 + 1)

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

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

**Parameters**

- **fft_data** – complex array from perform_fft
- **fft_len** – len of original array, must be even
- **data_len** – output array len

**Returns** restored data

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

**Parameters** **value** – input value

**Returns** nearest power of 2

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

**Parameters**

- **data** – input array
- **data_len** – must be even
- **sampling_rate** – sampling rate
- **window** – window function
- **psd_len** – output len (data_len / 2 + 1)

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

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

**Parameters**

- **data** – input array
- **data_len** –
- **psd_len** – output len (data_len / 2 + 1)

- **detrend_operation** – use DetrendOperations enum

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

calculate 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

**Returns** band power

static 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

**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

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

static std::pair<double*, double*> **get_custom_band_powers**(const BrainFlowArray<double, 2> &data, std::vector<std::pair<double, double>> bands, std::vector<int> channels, int sampling_rate, bool apply_filters)

calculate avg and stddev of BandPowers across all channels

**Parameters**

- **data** – input 2d array
- **bands** – input bands
- **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

**Returns** pair of float arrays with the same size as bands argument

static double **get_oxygen_level**(double *ppg_ir, double *ppg_red, int data_len, int sampling_rate, double coef1 = 1.5958422, double coef2 = -34.6596622, double coef3 = 112.6898759)

calculate oxygen level

**Parameters**

- **ppg_ir** – input 1d array

- **ppg_red** – input 1d array
- **data_len** – size of array
- **sampling_rate** – sampling rate

**Returns** oxygen level

static double **get_heart_rate**(double *ppg_ir, double *ppg_red, int data_len, int sampling_rate, int fft_size)
calculate heart rate

**Parameters**

- **ppg_ir** – input 1d array
- **ppg_red** – input 1d array
- **data_len** – size of array
- **sampling_rate** – sampling rate
- **fft_size** – recommended 8192

**Returns** heart rate

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

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

static double **calc_stddev**(double *data, int start_pos, int end_pos)
calc stddev

static double **get_railed_percentage**(double *data, int data_len, int gain)
calc railed percentage

static std::tuple<BrainFlowArray<double, 2>, BrainFlowArray<double, 2>, BrainFlowArray<double, 2>, BrainFlowArray<doubl

calculate ICA

**Parameters**

- **data** – input 2d array, rows are samples
- **num_components** – number of components to find
- **channels** – rows to use

**Returns** unmixed signal

static std::tuple<BrainFlowArray<double, 2>, BrainFlowArray<double, 2>, BrainFlowArray<double, 2>, BrainFlowArray<doubl

calculate ICA

**Parameters**

- **data** – input 2d array, rows are samples
- **num_components** – number of components to find

**Returns** unmixed signal

static std::string **get_version**()
get brainflow version

### 3.2.3 MLModel class

class **MLModel**
Calculates different metrics from raw data.

#### Public Functions

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

std::vector<double> **predict**(double *data, int data_len)
calculate metric from data

void **release**()
release classifier

#### Public Static Functions

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

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

static void **disable_ml_logger**()
disable ML loggers

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

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

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

static void **release_all**()
release all currently prepared classifiers

static std::string **get_version**()
get brainflow version

### 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
{
    NO_BOARD = -100, // only for internal usage
    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,
```

(continues on next page)

(continued from previous page)

```
    CALLIBRI_EEG_BOARD = 9,
    CALLIBRI_EMG_BOARD = 10,
    CALLIBRI_ECG_BOARD = 11,
    NOTION_1_BOARD = 13,
    NOTION_2_BOARD = 14,
    GFORCE_PRO_BOARD = 16,
    FREEEEG32_BOARD = 17,
    BRAINBIT_BLED_BOARD = 18,
    GFORCE_DUAL_BOARD = 19,
    GALEA_SERIAL_BOARD = 20,
    MUSE_S_BLED_BOARD = 21,
    MUSE_2_BLED_BOARD = 22,
    CROWN_BOARD = 23,
    ANT_NEURO_EE_410_BOARD = 24,
    ANT_NEURO_EE_411_BOARD = 25,
    ANT_NEURO_EE_430_BOARD = 26,
    ANT_NEURO_EE_211_BOARD = 27,
    ANT_NEURO_EE_212_BOARD = 28,
    ANT_NEURO_EE_213_BOARD = 29,
    ANT_NEURO_EE_214_BOARD = 30,
    ANT_NEURO_EE_215_BOARD = 31,
    ANT_NEURO_EE_221_BOARD = 32,
    ANT_NEURO_EE_222_BOARD = 33,
    ANT_NEURO_EE_223_BOARD = 34,
    ANT_NEURO_EE_224_BOARD = 35,
    ANT_NEURO_EE_225_BOARD = 36,
    ENOPHONE_BOARD = 37,
    MUSE_2_BOARD = 38,
    MUSE_S_BOARD = 39,
    BRAINALIVE_BOARD = 40,
    MUSE_2016_BOARD = 41,
    MUSE_2016_BLED_BOARD = 42,
    EXPLORE_4_CHAN_BOARD = 44,
    EXPLORE_8_CHAN_BOARD = 45,
    GANGLION_NATIVE_BOARD = 46,
    EMOTIBIT_BOARD = 47,
    GALEA_BOARD_V4 = 48,
    GALEA_SERIAL_BOARD_V4 = 49,
    NTL_WIFI_BOARD = 50,
    ANT_NEURO_EE_511_BOARD = 51,
    FREEEEG128_BOARD = 52,
    AAVAA_V3_BOARD = 53,
    EXPLORE_PLUS_8_CHAN_BOARD = 54,
    EXPLORE_PLUS_32_CHAN_BOARD = 55,
    // use it to iterate
    FIRST = PLAYBACK_FILE_BOARD,
    LAST = EXPLORE_PLUS_32_CHAN_BOARD
};

enum class IpProtocolTypes : int
{
    NO_IP_PROTOCOL = 0,
```

(continues on next page)

(continued from previous page)

```
    UDP = 1,
    TCP = 2
};

enum class FilterTypes : int
{
    BUTTERWORTH = 0,
    CHEBYSHEV_TYPE_1 = 1,
    BESSEL = 2,
    BUTTERWORTH_ZERO_PHASE = 3,
    CHEBYSHEV_TYPE_1_ZERO_PHASE = 4,
    BESSEL_ZERO_PHASE = 5
};

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

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

enum class DetrendOperations : int
{
    NO_DETREND = 0,
    CONSTANT = 1,
    LINEAR = 2
};

enum class BrainFlowMetrics : int
{
    MINDFULNESS = 0,
    RESTFULNESS = 1,
    USER_DEFINED = 2
};

enum class BrainFlowClassifiers : int
{
    DEFAULT_CLASSIFIER = 0,
    DYN_LIB_CLASSIFIER = 1,
    ONNX_CLASSIFIER = 2
};

enum class BrainFlowPresets : int
{
```

(continues on next page)

(continued from previous page)

```
    DEFAULT_PRESET = 0,
    AUXILIARY_PRESET = 1,
    ANCILLARY_PRESET = 2
};

enum class LogLevels : int
{
    LEVEL_TRACE = 0,
    LEVEL_DEBUG = 1,
    LEVEL_INFO = 2,
    LEVEL_WARN = 3,
    LEVEL_ERROR = 4,
    LEVEL_CRITICAL = 5,
    LEVEL_OFF = 6
};

enum class NoiseTypes : int
{
    FIFTY = 0,
    SIXTY = 1,
    FIFTY_AND_SIXTY = 2
};

enum class WaveletDenoisingTypes : int
{
    VISUSHRINK = 0,
    SURESHRINK = 1
};

enum class ThresholdTypes : int
{
    SOFT = 0,
    HARD = 1
};

enum class WaveletExtensionTypes : int
{
    SYMMETRIC = 0,
    PERIODIC = 1
};

enum class NoiseEstimationLevelTypes : int
{
    FIRST_LEVEL = 0,
    ALL_LEVELS = 1
};

enum class WaveletTypes : int
{
    HAAR = 0,
    DB1 = 1,
    DB2 = 2,
```

(continues on next page)

(continued from previous page)

```
    DB3 = 3,
    DB4 = 4,
    DB5 = 5,
    DB6 = 6,
    DB7 = 7,
    DB8 = 8,
    DB9 = 9,
    DB10 = 10,
    DB11 = 11,
    DB12 = 12,
    DB13 = 13,
    DB14 = 14,
    DB15 = 15,
    BIOR1_1 = 16,
    BIOR1_3 = 17,
    BIOR1_5 = 18,
    BIOR2_2 = 19,
    BIOR2_4 = 20,
    BIOR2_6 = 21,
    BIOR2_8 = 22,
    BIOR3_1 = 23,
    BIOR3_3 = 24,
    BIOR3_5 = 25,
    BIOR3_7 = 26,
    BIOR3_9 = 27,
    BIOR4_4 = 28,
    BIOR5_5 = 29,
    BIOR6_8 = 30,
    COIF1 = 31,
    COIF2 = 32,
    COIF3 = 33,
    COIF4 = 34,
    COIF5 = 35,
    SYM2 = 36,
    SYM3 = 37,
    SYM4 = 38,
    SYM5 = 39,
    SYM6 = 40,
    SYM7 = 41,
    SYM8 = 42,
    SYM9 = 43,
    SYM10 = 44,
    // to iterate and check sizes
    FIRST_WAVELET = HAAR,
    LAST_WAVELET = SYM10
};
```

## 3.3 Java API Reference

Content of Brainflow Package:

enum **AggOperations**
enum to store all supported aggregation operations

#### Public Functions

inline int **get_code**()

```
inline  brainflow::AggOperations (final int code)
```

#### Public Members

```
brainflow::MEAN   =(0)
brainflow::MEDIAN   =(1)
brainflow::EACH   =(2)
```

#### Public Static Functions

```
static inline String brainflow::string_from_code (final int code)
static inline AggOperations brainflow::from_code (final int code)
static inline  brainflow::[static initializer]
```

#### Private Members

```
final int brainflow::agg_operation
```

#### Private Static Attributes

```
static final Map< Integer, AggOperations > brainflow::
ao_map   = new HashMap<Integer, AggOperations> ()
```

class **BoardDescr**

enum **BoardIds**
enum to store all supported boards

### Public Functions

inline int **get_code**()

**inline brainflow::BoardIds (final int code)**

### Public Members

**brainflow::NO_BOARD =(-100)**

**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::NOTION_1_BOARD =(13)**

**brainflow::NOTION_2_BOARD =(14)**

**brainflow::GFORCE_PRO_BOARD =(16)**

**brainflow::FREEEEG32_BOARD =(17)**

**brainflow::BRAINBIT_BLED_BOARD =(18)**

**brainflow::GFORCE_DUAL_BOARD =(19)**

**brainflow::GALEA_SERIAL_BOARD =(20)**

**brainflow::MUSE_S_BLED_BOARD =(21)**

**brainflow::MUSE_2_BLED_BOARD =(22)**

**brainflow::CROWN_BOARD =(23)**

**brainflow::ANT_NEURO_EE_410_BOARD =(24)**

**brainflow::ANT_NEURO_EE_411_BOARD =(25)**

**brainflow::ANT_NEURO_EE_430_BOARD =(26)**

brainflow::ANT_NEURO_EE_211_BOARD =(27)

brainflow::ANT_NEURO_EE_212_BOARD =(28)

brainflow::ANT_NEURO_EE_213_BOARD =(29)

brainflow::ANT_NEURO_EE_214_BOARD =(30)

brainflow::ANT_NEURO_EE_215_BOARD =(31)

brainflow::ANT_NEURO_EE_221_BOARD =(32)

brainflow::ANT_NEURO_EE_222_BOARD =(33)

brainflow::ANT_NEURO_EE_223_BOARD =(34)

brainflow::ANT_NEURO_EE_224_BOARD =(35)

brainflow::ANT_NEURO_EE_225_BOARD =(36)

brainflow::ENOPHONE_BOARD =(37)

brainflow::MUSE_2_BOARD =(38)

brainflow::MUSE_S_BOARD =(39)

brainflow::BRAINALIVE_BOARD =(40)

brainflow::MUSE_2016_BOARD =(41)

brainflow::MUSE_2016_BLED_BOARD =(42)

brainflow::EXPLORE_4_CHAN_BOARD =(44)

brainflow::EXPLORE_8_CHAN_BOARD =(45)

brainflow::GANGLION_NATIVE_BOARD =(46)

brainflow::EMOTIBIT_BOARD =(47)

brainflow::GALEA_BOARD_V4 =(48)

brainflow::GALEA_SERIAL_BOARD_V4 =(49)

brainflow::NTL_WIFI_BOARD =(50)

brainflow::ANT_NEURO_EE_511_BOARD =(51)

brainflow::FREEEEG128_BOARD =(52)

brainflow::AAVAA_V3_BOARD =(53)

brainflow::EXPLORE_PLUS_8_CHAN_BOARD =(54)

brainflow::EXPLORE_PLUS_32_CHAN_BOARD =(55)

### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline BoardIds brainflow::from_code (final int code)

static inline  brainflow::[static initializer]

### Private Members

final int brainflow::board_id

### Private Static Attributes

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

class brainflow::brainflow::**BoardShim**
BoardShim class to communicate with a board

### Public Functions

inline **BoardShim**(int board_id, *BrainFlowInputParams* params)
Create BoardShim object

inline **BoardShim**(*BoardIds* board_id, *BrainFlowInputParams* params)
Create BoardShim object

inline void **prepare_session**()
prepare steaming session, allocate resources

inline int **get_board_id**()
Get Board Id, can be different than provided (playback or streaming board)

inline void **add_streamer**(String streamer, int preset)
add streamer

inline void **delete_streamer**(String streamer, int preset)
delete streamer

inline String **config_board**(String config)
send string to a board, use this method carefully and only if you understand what you are doing

inline void config_board_with_bytes (byte[] bytes)
send string to a board, dont use it

inline 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”

inline void **start_stream**()
start streaming thread, store data in internal ringbuffer

inline void **start_stream**(int buffer_size)
start streaming thread, store data in internal ringbuffer

inline void **stop_stream**()
stop streaming thread

inline void **release_session**()
release all resources

inline int **get_board_data_count**(*BrainFlowPresets* preset)
get number of packages in ringbuffer

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

inline void **insert_marker**(double value, *BrainFlowPresets* preset)
insert marker to data stream

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

inline boolean **is_prepared**()
check session status

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

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

**inline double[][] get_board_data (BrainFlowPresets preset)**
get all data from ringbuffer and flush it

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

### Public Members

int **board_id**
BrainFlow's board id

### Public Static Functions

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

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

static inline void **disable_board_logger**()
disable BrainFlow logger

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

static inline void **release_all_sessions**()
release all prepared sessions

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

static inline void **set_log_level**(*LogLevels* log_level)
set log level

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

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

static inline int **get_sampling_rate**(int board_id, *BrainFlowPresets* preset)
get sampling rate for this board

static inline int **get_sampling_rate**(*BoardIds* board_id, *BrainFlowPresets* preset)
get sampling rate for this board

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

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

static inline int **get_timestamp_channel**(int board_id, *BrainFlowPresets* preset)
get row index in returned by *get_board_data()* 2d array which contains timestamps

static inline int **get_timestamp_channel**(*BoardIds* board_id, *BrainFlowPresets* preset)
get row index in returned by *get_board_data()* 2d array which contains timestamps

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

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

static inline int **get_marker_channel**(int board_id, *BrainFlowPresets* preset)
get row index in returned by *get_board_data()* 2d array which contains markers

static inline int **get_marker_channel**(*BoardIds* board_id, *BrainFlowPresets* preset)
get row index in returned by *get_board_data()* 2d array which contains markers

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

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

static inline int **get_num_rows**(int board_id, *BrainFlowPresets* preset)
get number of rows in returned by *get_board_data()* 2d array

static inline int **get_num_rows**(*BoardIds* board_id, *BrainFlowPresets* preset)
get number of rows in returned by *get_board_data()* 2d array

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

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

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

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

static inline int **get_package_num_channel**(*BoardIds* board_id, *BrainFlowPresets* preset)
get row index in returned by *get_board_data()* 2d array which contains package nums

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

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

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

static inline int **get_battery_channel**(*BoardIds* board_id, *BrainFlowPresets* preset)
get row index in returned by *get_board_data()* 2d array which contains battery level

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

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

**static inline BrainFlowPresets[] get_board_presets (int board_id)**
Get supported presets for this device

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

**static inline String[] get_eeg_names (BoardIds board_id, BrainFlowPresets preset)**
Get names of EEG electrodes in 10-20 system. Only if electrodes have freezed locations

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

**static inline static< T > T get_board_descr (Class< T > type, int board_id, BrainFlowPresets preset)**
Get board description

**static inline static< T > T get_board_descr (Class< T > type, int board_id)**
Get board description

**static inline static< T > T get_board_descr (Class< T > type, BoardIds board_id, BrainFlowPresets preset)**
Get board description

**static inline static< T > T get_board_descr (Class< T > type, BoardIds board_id)**
Get board description

static inline String **get_device_name**(int board_id, *BrainFlowPresets* preset)
Get device name

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

static inline String **get_device_name**(*BoardIds* board_id, *BrainFlowPresets* preset)
Get device name

static inline String **get_device_name**(*BoardIds* board_id)
Get device name

static inline String **get_version**()
Get version

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

**static inline 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\EMG... and return the same array

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

**static inline int[] get_eeg_channels (BoardIds 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\EMG... and return the same array

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

**static inline 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\EMG... and return the same array

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

**static inline int[] get_emg_channels (BoardIds 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\EMG... and return the same array

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

**static inline 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\EMG... and return the same array

**static inline int[] get_ecg_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain ECG data, for some boards we can

not split EEG\EMG... and return the same array

**static inline int[] get_ecg_channels (BoardIds 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\EMG... and return the same array

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

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

**static inline int[] get_temperature_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain temperature data

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

**static inline int[] get_magnetometer_channels (int board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain magnetometer data

**static inline int[] get_magnetometer_channels (int board_id)**
get row indices in returned by *get_board_data()* 2d array which contain magnetometer data

**static inline int[] get_magnetometer_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain magnetometer data

**static inline int[] get_magnetometer_channels (BoardIds board_id)**
get row indices in returned by *get_board_data()* 2d array which contain magnetometer data

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

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

**static inline int[] get_resistance_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain resistance data

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

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

**static inline 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\EMG... and return the same array

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

**static inline int[] get_eog_channels (BoardIds 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\EMG... and return the same array

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

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

**static inline int[] get_exg_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain EXG data

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

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

**static inline 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\EMG... and return the same array

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

**static inline int[] get_eda_channels (BoardIds 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\EMG... and return the same array

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

**static inline 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\EMG... and return the same array

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

**static inline int[] get_ppg_channels (BoardIds 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\EMG... and return the same array

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

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

**static inline int[] get_accel_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain accel data

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

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

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

**static inline int[] get_analog_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain analog data

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

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

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

**static inline int[] get_gyro_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain gyro data

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

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

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

**static inline int[] get_other_channels (BoardIds board_id, BrainFlowPresets preset)**
get row indices in returned by *get_board_data()* 2d array which contain other data

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

enum **BrainFlowClassifiers**

### Public Functions

inline int **get_code**()

**inline brainflow::BrainFlowClassifiers (final int code)**

### Public Members

**brainflow::DEFAULT_CLASSIFIER =(0)**

**brainflow::DYN_LIB_CLASSIFIER =(1)**

**brainflow::ONNX_CLASSIFIER =(2)**

### Public Static Functions

**static inline String brainflow::string_from_code (final int code)**

**static inline BrainFlowClassifiers brainflow::from_code (final int code)**

**static inline brainflow::[static initializer]**

### Private Members

**final int brainflow::protocol**

### Private Static Attributes

**static final Map< Integer, BrainFlowClassifiers > brainflow::**
**cl_map = new HashMap<Integer, BrainFlowClassifiers> ()**

class brainflow::brainflow::**BrainFlowError** : public Exception
BrainFlowError exception to notify about errors

### Public Members

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

enum **BrainFlowExitCode**

### Public Functions

inline int **get_code**()

**inline  brainflow::BrainFlowExitCode (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

```
static inline String brainflow::string_from_code (final int code)
static inline BrainFlowExitCode brainflow::from_code (final int code)
static inline  brainflow::[static initializer]
```

### Private Members

```
final int brainflow::exit_code
```

### Private Static Attributes

```
static final Map< Integer, BrainFlowExitCode > brainflow::
ec_map   = new HashMap<Integer, BrainFlowExitCode> ()
```

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

enum **BrainFlowMetrics**

### Public Functions

inline int **get_code**()

```
inline  brainflow::BrainFlowMetrics (final int code)
```

### Public Members

```
brainflow::MINDFULNESS   =(0)
brainflow::RESTFULNESS   =(1)
brainflow::USER_DEFINED   =(2)
```

### Public Static Functions

```
static inline String brainflow::string_from_code (final int code)
static inline BrainFlowMetrics brainflow::from_code (final int code)
static inline  brainflow::[static initializer]
```

### Private Members

final int brainflow::protocol

### Private Static Attributes

static final Map< Integer, BrainFlowMetrics > brainflow::
metr_map = new HashMap<Integer, BrainFlowMetrics> ()

class **BrainFlowModelParams**
describe model parameters

enum **BrainFlowPresets**
enum to store all supported presets

### Public Functions

inline int **get_code**()

inline brainflow::BrainFlowPresets (final int code)

### Public Members

brainflow::DEFAULT_PRESET =(0)

brainflow::AUXILIARY_PRESET =(1)

brainflow::ANCILLARY_PRESET =(2)

### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline BrainFlowPresets brainflow::from_code (final int code)

static inline brainflow::[static initializer]

### Private Members

final int brainflow::preset

### Private Static Attributes

```
static final Map< Integer, BrainFlowPresets > brainflow::
my_map   = new HashMap<Integer, BrainFlowPresets> ()
```

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

### Public Static Functions

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

static inline String **get_version**()
Get version

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

static inline void **disable_data_logger**()
disable Data logger

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

**static inline double calc_stddev (double[] data, int start_pos, int end_pos)**
calc stddev

**static inline double get_oxygen_level (double[] ppg_ir, double[] ppg_red, int sampling_rate, double coef1, double coef2, double coef3)**
get oxygen level

**static inline double get_oxygen_level (double[] ppg_ir, double[] ppg_red, int sampling_rate)**
get oxygen level

**static inline double get_heart_rate (double[] ppg_ir, double[] ppg_red, int sampling_rate, int fft_size)**
get heart rate

**static inline double get_railed_percentage (double[] data, int len, int gain)**
get railed percentage

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

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

static inline void **set_log_level**(*LogLevels* log_level)
set log level

**static inline void perform_lowpass (double[] data, int sampling_rate, double cutoff, int order, int filter_type, double ripple)**
perform lowpass filter in-place

**static inline void perform_lowpass (double[] data, int sampling_rate, double cutoff, int order, FilterTypes filter_type, double ripple)**
perform lowpass filter in-place

**static inline void perform_highpass (double[] data, int sampling_rate, double cutoff, int order, int filter_type, double ripple)**
perform highpass filter in-place

**static inline void perform_highpass (double[] data, int sampling_rate, double cutoff, int order, FilterTypes filter_type, double ripple)**
perform highpass filter in-place

**static inline void perform_bandpass (double[] data, int sampling_rate, double start_freq, double stop_freq, int order, int filter_type, double ripple)**
perform bandpass filter in-place

**static inline void perform_bandpass (double[] data, int sampling_rate, double start_freq, double stop_freq, int order, FilterTypes filter_type, double ripple)**
perform bandpass filter in-place

**static inline void perform_bandstop (double[] data, int sampling_rate, double start_freq, double stop_freq, int order, int filter_type, double ripple)**
perform bandstop filter in-place

**static inline void perform_bandstop (double[] data, int sampling_rate, double start_freq, double stop_freq, int order, FilterTypes filter_type, double ripple)**
perform bandstop filter in-place

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

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

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

**static inline void detrend (double[] data, DetrendOperations operation)**
subtract trend from data in-place

**static inline 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

**static inline double[] restore_data_from_wavelet_detailed_coeffs (double[] data, int wavelet, int decomposition_level, int level_to_restore)**
restore data from a single wavelet coeff

**static inline double[] restore_data_from_wavelet_detailed_coeffs (double[] data, WaveletTypes wavelet, int decomposition_level, int level_to_restore)**
restore data from a single wavelet coeff

**static inline double[] detect_peaks_z_score (double[] data, int lag, double threshold, double influence)**
peak detection using z score algorithm

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

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

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

**static inline void perform_wavelet_denoising (double[] data, int wavelet, int decomposition_level, int wavelet_denoising, int threshold, int extenstion_type, int noise_level)**
perform wavelet based denoising in-place

**static inline void perform_wavelet_denoising (double[] data, WaveletTypes wavelet, int decomposition_level, WaveletDenoisingTypes wavelet_denoising, ThresholdTypes threshold, WaveletExtensionTypes extenstion_type, NoiseEstimationLevelTypes noise_level)**
perform wavelet based denoising in-place

**static inline Pair< double[], int[]> perform_wavelet_transform (double[] data, int wavelet, int decomposition_level, int extension)**
perform wavelet transform

**static inline Pair< double[], int[]> perform_wavelet_transform (double[] data, WaveletTypes wavelet, int decomposition_level, WaveletExtensionTypes extension)**
perform wavelet transform

**static inline double[] perform_inverse_wavelet_transform (Pair< double[], int[]> wavelet_output, int original_data_len, int wavelet, int decomposition_level, int extension)**
perform inverse wavelet transform

**static inline double[] perform_inverse_wavelet_transform (Pair< double[], int[]> wavelet_output, int original_data_len, WaveletTypes wavelet, int decomposition_level, WaveletExtensionTypes extension)**
perform inverse wavelet transform

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

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

**Parameters**

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

**Returns** array of the size specified in window_len

`static inline double[] get_window (WindowOperations window, int window_len)`
perform data windowing

**Parameters**

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

**Returns** array of the size specified in window_len

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

**Parameters**

- `data` – data for fft transform
- `start_pos` – starting position to calc fft
- `end_pos` – end position to calc fft, total_len must be even
- `window` – window function

**Returns** array of complex values with size N / 2 + 1

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

**Parameters**

- `data` – data for fft transform
- `start_pos` – starting position to calc fft
- `end_pos` – end position to calc fft, total_len must be even
- `window` – window function

**Returns** array of complex values with size N / 2 + 1

`static inline double[] perform_ifft (Complex[] data)`
perform inverse fft

**Parameters** `data` – data from fft transform(array of complex values)

**Returns** restored data

`static inline Pair< double[], double[]> get_avg_band_powers (double[][] data, int[] channels, int sampling_rate, boolean apply_filters)`
calc average and stddev of band powers across all channels

**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

**Returns** pair of avgs and stddevs for bandpowers

`static inline Pair< double[], double[]> get_custom_band_powers (double[][] data, List< Pair< Double, Double >> bands, int[] channels, int sampling_rate, boolean apply_filters)`

calc average and stddev of band powers across all channels

**Parameters**

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

**Returns** pair of avgs and stddevs for bandpowers

`static inline List< double[][]> perform_ica (double[][] data, int num_components)`

Calculates ICA

**Parameters**

- **data** –
- **num_components** –

**Throws** *BrainFlowError* –

**Returns**

`static inline List< double[][]> perform_ica (double[][] data, int num_components, int[] channels)`

calculates ICA

**Parameters**

- **data** –
- **num_components** –
- **channels** –

**Throws** *BrainFlowError* –

**Returns**

`static inline Pair< double[], double[]> get_psd (double[] data, int start_pos, int end_pos, int sampling_rate, int window)`

get PSD

**Parameters**

- **data** – data to process
- **start_pos** – starting position to calc PSD
- **end_pos** – end position to calc PSD, total_len must be even
- **sampling_rate** – sampling rate
- **window** – window function

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

`static inline Pair< double[], double[]> get_psd (double[] data, int start_pos, int end_pos, int sampling_rate, WindowOperations window)`
get PSD

**Parameters**

- **data** – data to process
- **start_pos** – starting position to calc PSD
- **end_pos** – end position to calc PSD, total_len must be even
- **sampling_rate** – sampling rate
- **window** – window function

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

`static inline Pair< double[], double[]> get_psd_welch (double[] data, int nfft, int overlap, int sampling_rate, int window)`
get PSD using Welch Method

**Parameters**

- **data** – data to process
- **nfft** – size of FFT, must be even
- **overlap** – overlap between FFT Windows, must be between 0 and nfft
- **sampling_rate** – sampling rate
- **window** – window function

**Returns** pair of ampl and freq arrays

`static inline Pair< double[], double[]> get_psd_welch (double[] data, int nfft, int overlap, int sampling_rate, WindowOperations window)`
get PSD using Welch Method

**Parameters**

- **data** – data to process
- **nfft** – size of FFT, must be even
- **overlap** – overlap between FFT Windows, must be between 0 and nfft
- **sampling_rate** – sampling rate
- **window** – window function

**Returns** pair of ampl and freq arrays

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

**Parameters**

- **psd** – PSD from get_psd or get_log_psd
- **freq_start** – lowest frequency of band
- **freq_end** – highest frequency of band

**Returns** band power

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

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

**static inline 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

inline int **get_code**()

**inline  brainflow::DetrendOperations (final int code)**

### Public Members

**brainflow::NO_DETREND  =(0)**

**brainflow::CONSTANT  =(1)**

**brainflow::LINEAR  =(2)**

### Public Static Functions

**static inline String brainflow::string_from_code (final int code)**

**static inline DetrendOperations brainflow::from_code (final int code)**

**static inline  brainflow::[static initializer]**

### Private Members

final int brainflow::detrend_operation

### Private Static Attributes

static final Map< Integer, DetrendOperations > brainflow::
dt_map = new HashMap<Integer, DetrendOperations> ()

enum FilterTypes
enum to store all possible filter types

### Public Functions

inline int get_code()

inline brainflow::FilterTypes (final int code)

### Public Members

brainflow::BUTTERWORTH =(0)

brainflow::CHEBYSHEV_TYPE_1 =(1)

brainflow::BESSEL =(2)

brainflow::BUTTERWORTH_ZERO_PHASE =(3)

brainflow::CHEBYSHEV_TYPE_1_ZERO_PHASE =(4)

brainflow::BESSEL_ZERO_PHASE =(5)

### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline FilterTypes brainflow::from_code (final int code)

static inline brainflow::[static initializer]

### Private Members

final int brainflow::filter_type

#### Private Static Attributes

static final Map< Integer, FilterTypes > brainflow::ft_map = new HashMap<Integer, FilterTypes> ()

enum IpProtocolTypes

#### Public Functions

inline int get_code()

inline brainflow::IpProtocolTypes (final int code)

#### Public Members

brainflow::NO_IP_PROTOCOL =(0)

brainflow::UDP =(1)

brainflow::TCP =(2)

#### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline IpProtocolTypes brainflow::from_code (final int code)

static inline brainflow::[static initializer]

#### Private Members

final int brainflow::protocol

#### Private Static Attributes

static final Map< Integer, IpProtocolTypes > brainflow::
ip_map = new HashMap<Integer, IpProtocolTypes> ()

class JarHelper

enum LogLevels

### Public Functions

inline int **get_code**()

**inline  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

**static inline String brainflow::string_from_code (final int code)**

**static inline LogLevels brainflow::from_code (final int code)**

**static inline  brainflow::[static initializer]**

### Private Members

**final int brainflow::log_level**

### Private Static Attributes

**static final Map< Integer, LogLevels > brainflow::ll_map  = new HashMap<Integer, LogLevels> ()**

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

### Public Functions

inline **MLModel**(*BrainFlowModelParams* params)
Create MLModel object

inline void **prepare**()
Prepare classifier

**Throws** *BrainFlowError* –

inline void **release**()
Release classifier

**Throws** *BrainFlowError* –

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

**Throws** *BrainFlowError* –

### Public Static Functions

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

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

static inline void **disable_ml_logger**()
disable BrainFlow logger

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

static inline String **get_version**()
Get version

static inline void **release_all**()
release all classifiers

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

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

static inline void **set_log_level**(*LogLevels* log_level)
set log level

enum **NoiseEstimationLevelTypes**
enum to store all possible Noise Level Types

### Public Functions

inline int **get_code**()

**inline brainflow::NoiseEstimationLevelTypes (final int code)**

### Public Members

**brainflow::FIRST_LEVEL =(0)**

**brainflow::ALL_LEVELS =(1)**

### Public Static Functions

**static inline String brainflow::string_from_code (final int code)**

**static inline NoiseEstimationLevelTypes brainflow::from_code (final int code)**

**static inline brainflow::[static initializer]**

### Private Members

**final int brainflow::value**

### Private Static Attributes

```
static final Map< Integer, NoiseEstimationLevelTypes > brainflow::
map = new HashMap<Integer, NoiseEstimationLevelTypes> ()
```

enum **NoiseTypes**

enum to store all supported noise types

### Public Functions

inline int **get_code**()

**inline brainflow::NoiseTypes (final int code)**

### Public Members

brainflow::FIFTY =(0)

brainflow::SIXTY =(1)

brainflow::FIFTY_AND_SIXTY =(2)

### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline NoiseTypes brainflow::from_code (final int code)

static inline brainflow::[static initializer]

### Private Members

final int brainflow::noise_type

### Private Static Attributes

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

enum **ThresholdTypes**
enum to store all possible Threshold Types

### Public Functions

inline int **get_code**()

inline brainflow::ThresholdTypes (final int code)

### Public Members

brainflow::SOFT =(0)

brainflow::HARD =(1)

### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline ThresholdTypes brainflow::from_code (final int code)

static inline  brainflow::[static initializer]

### Private Members

final int brainflow::value

### Private Static Attributes

static final Map< Integer, ThresholdTypes > brainflow::map  = new HashMap<Integer, ThresholdTypes> ()

enum WaveletDenoisingTypes
enum to store all possible Wavelet Denoising Types

### Public Functions

inline int get_code()

inline  brainflow::WaveletDenoisingTypes (final int code)

### Public Members

brainflow::VISUSHRINK  =(0)

brainflow::SURESHRINK  =(1)

### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline WaveletDenoisingTypes brainflow::from_code (final int code)

static inline  brainflow::[static initializer]

### Private Members

final int brainflow::value

### Private Static Attributes

static final Map< Integer, WaveletDenoisingTypes > brainflow::
map = new HashMap<Integer, WaveletDenoisingTypes> ()

enum **WaveletExtensionTypes**
enum to store all possible Wavelet Extenstion Types

### Public Functions

inline int **get_code**()

inline brainflow::WaveletExtensionTypes (final int code)

### Public Members

brainflow::SYMMETRIC =(0)

brainflow::PERIODIC =(1)

### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline WaveletExtensionTypes brainflow::from_code (final int code)

static inline brainflow::[static initializer]

### Private Members

final int brainflow::value

### Private Static Attributes

static final Map< Integer, WaveletExtensionTypes > brainflow::
map = new HashMap<Integer, WaveletExtensionTypes> ()

enum **WaveletTypes**
enum to store all possible Wavelet Types

### Public Functions

inline int **get_code**()

**inline brainflow::WaveletTypes (final int code)**

### Public Members

**brainflow::HAAR =(0)**

**brainflow::DB1 =(1)**

**brainflow::DB2 =(2)**

**brainflow::DB3 =(3)**

**brainflow::DB4 =(4)**

**brainflow::DB5 =(5)**

**brainflow::DB6 =(6)**

**brainflow::DB7 =(7)**

**brainflow::DB8 =(8)**

**brainflow::DB9 =(9)**

**brainflow::DB10 =(10)**

**brainflow::DB11 =(11)**

**brainflow::DB12 =(12)**

**brainflow::DB13 =(13)**

**brainflow::DB14 =(14)**

**brainflow::DB15 =(15)**

**brainflow::BIOR1_1 =(16)**

**brainflow::BIOR1_3 =(17)**

**brainflow::BIOR1_5 =(18)**

**brainflow::BIOR2_2 =(19)**

**brainflow::BIOR2_4 =(20)**

**brainflow::BIOR2_6 =(21)**

**brainflow::BIOR2_8 =(22)**

**brainflow::BIOR3_1 =(23)**

**brainflow::BIOR3_3 =(24)**

**brainflow::BIOR3_5 =(25)**

**brainflow::BIOR3_7 =(26)**

**brainflow::BIOR3_9 =(27)**

**brainflow::BIOR4_4 =(28)**

brainflow::BIOR5_5 =(29)

brainflow::BIOR6_8 =(30)

brainflow::COIF1 =(31)

brainflow::COIF2 =(32)

brainflow::COIF3 =(33)

brainflow::COIF4 =(34)

brainflow::COIF5 =(35)

brainflow::SYM2 =(36)

brainflow::SYM3 =(37)

brainflow::SYM4 =(38)

brainflow::SYM5 =(39)

brainflow::SYM6 =(40)

brainflow::SYM7 =(41)

brainflow::SYM8 =(42)

brainflow::SYM9 =(43)

brainflow::SYM10 =(44)

#### Public Static Functions

static inline String brainflow::string_from_code (final int code)

static inline WaveletTypes brainflow::from_code (final int code)

static inline brainflow::[static initializer]

#### Private Members

final int brainflow::value

#### Private Static Attributes

static final Map< Integer, WaveletTypes > brainflow::map = new HashMap<Integer, WaveletTypes> ()

enum WindowOperations

#### Public Functions

inline int **get_code**()

**inline  brainflow::WindowOperations (final int code)**

#### Public Members

**brainflow::NO_WINDOW  =(0)**

**brainflow::HANNING  =(1)**

**brainflow::HAMMING  =(2)**

**brainflow::BLACKMAN_HARRIS  =(3)**

#### Public Static Functions

**static inline String brainflow::string_from_code (final int code)**

**static inline WindowOperations brainflow::from_code (final int code)**

**static inline  brainflow::[static initializer]**

#### Private Members

**final int brainflow::window**

#### Private Static Attributes

**static final Map< Integer, WindowOperations > brainflow::**
**window_map  = new HashMap<Integer, WindowOperations> ()**

## 3.4 C# API Reference

Content of brainflow namespace:

**brainflow.LogLevels**
*Values:*

enumerator **LEVEL_TRACE**

enumerator **LEVEL_DEBUG**

enumerator **LEVEL_INFO**

enumerator **LEVEL_WARN**

enumerator **LEVEL_ERROR**

enumerator LEVEL_CRITICAL

enumerator LEVEL_OFF

brainflow.IpProtocolTypes
*Values:*

enumerator NO_IP_PROTOCOL

enumerator UDP

enumerator TCP

brainflow.BrainFlowPresets
*Values:*

enumerator DEFAULT_PRESET

enumerator AUXILIARY_PRESET

enumerator ANCILLARY_PRESET

brainflow.BrainFlowExitCodes
*Values:*

enumerator STATUS_OK

enumerator PORT_ALREADY_OPEN_ERROR

enumerator UNABLE_TO_OPEN_PORT_ERROR

enumerator SET_PORT_ERROR

enumerator BOARD_WRITE_ERROR

enumerator INCOMMING_MSG_ERROR

enumerator INITIAL_MSG_ERROR

enumerator BOARD_NOT_READY_ERROR

enumerator STREAM_ALREADY_RUN_ERROR

enumerator INVALID_BUFFER_SIZE_ERROR

enumerator STREAM_THREAD_ERROR

enumerator STREAM_THREAD_IS_NOT_RUNNING

enumerator EMPTY_BUFFER_ERROR

enumerator INVALID_ARGUMENTS_ERROR

enumerator UNSUPPORTED_BOARD_ERROR

enumerator BOARD_NOT_CREATED_ERROR

enumerator ANOTHER_BOARD_IS_CREATED_ERROR

enumerator GENERAL_ERROR

enumerator SYNC_TIMEOUT_ERROR

enumerator JSON_NOT_FOUND_ERROR

enumerator NO_SUCH_DATA_IN_JSON_ERROR

enumerator **CLASSIFIER_IS_NOT_PREPARED_ERROR**

enumerator **ANOTHER_CLASSIFIER_IS_PREPARED_ERROR**

enumerator **UNSUPPORTED_CLASSIFIER_AND_METRIC_COMBINATION_ERROR**

**brainflow.BoardIds**

*Values:*

enumerator **NO_BOARD**

enumerator **PLAYBACK_FILE_BOARD**

enumerator **STREAMING_BOARD**

enumerator **SYNTHETIC_BOARD**

enumerator **CYTON_BOARD**

enumerator **GANGLION_BOARD**

enumerator **CYTON_DAISY_BOARD**

enumerator **GALEA_BOARD**

enumerator **GANGLION_WIFI_BOARD**

enumerator **CYTON_WIFI_BOARD**

enumerator **CYTON_DAISY_WIFI_BOARD**

enumerator **BRAINBIT_BOARD**

enumerator **UNICORN_BOARD**

enumerator **CALLIBRI_EEG_BOARD**

enumerator **CALLIBRI_EMG_BOARD**

enumerator **CALLIBRI_ECG_BOARD**

enumerator **NOTION_1_BOARD**

enumerator **NOTION_2_BOARD**

enumerator **GFORCE_PRO_BOARD**

enumerator **FREEEEG32_BOARD**

enumerator **BRAINBIT_BLED_BOARD**

enumerator **GFORCE_DUAL_BOARD**

enumerator **GALEA_SERIAL_BOARD**

enumerator **MUSE_S_BLED_BOARD**

enumerator **MUSE_2_BLED_BOARD**

enumerator **CROWN_BOARD**

enumerator **ANT_NEURO_EE_410_BOARD**

enumerator **ANT_NEURO_EE_411_BOARD**

enumerator **ANT_NEURO_EE_430_BOARD**

enumerator **ANT_NEURO_EE_211_BOARD**

enumerator **ANT_NEURO_EE_212_BOARD**

enumerator **ANT_NEURO_EE_213_BOARD**

enumerator **ANT_NEURO_EE_214_BOARD**

enumerator **ANT_NEURO_EE_215_BOARD**

enumerator **ANT_NEURO_EE_221_BOARD**

enumerator **ANT_NEURO_EE_222_BOARD**

enumerator **ANT_NEURO_EE_223_BOARD**

enumerator **ANT_NEURO_EE_224_BOARD**

enumerator **ANT_NEURO_EE_225_BOARD**

enumerator **ENOPHONE_BOARD**

enumerator **MUSE_2_BOARD**

enumerator **MUSE_S_BOARD**

enumerator **BRAINALIVE_BOARD**

enumerator **MUSE_2016_BOARD**

enumerator **MUSE_2016_BLED_BOARD**

enumerator **EXPLORE_4_CHAN_BOARD**

enumerator **EXPLORE_8_CHAN_BOARD**

enumerator **GANGLION_NATIVE_BOARD**

enumerator **EMOTIBIT_BOARD**

enumerator **GALEA_BOARD_V4**

enumerator **GALEA_SERIAL_BOARD_V4**

enumerator **NTL_WIFI_BOARD**

enumerator **ANT_NEURO_EE_511_BOARD**

enumerator **FREEEEG128_BOARD**

enumerator **AAVAA_V3_BOARD**

enumerator **EXPLORE_PLUS_8_CHAN_BOARD**

enumerator **EXPLORE_PLUS_32_CHAN_BOARD**

**brainflow.FilterTypes**

*Values:*

enumerator **BUTTERWORTH**

enumerator **CHEBYSHEV_TYPE_1**

enumerator **BESSEL**

enumerator **BUTTERWORTH_ZERO_PHASE**

enumerator **CHEBYSHEV_TYPE_1_ZERO_PHASE**

enumerator **BESSEL_ZERO_PHASE**

**brainflow.AggOperations**

*Values:*

enumerator **MEAN**

enumerator **MEDIAN**

enumerator **EACH**

**brainflow.WindowOperations**
*Values:*

enumerator **NO_WINDOW**

enumerator **HANNING**

enumerator **HAMMING**

enumerator **BLACKMAN_HARRIS**

**brainflow.DetrendOperations**
*Values:*

enumerator **NO_DETREND**

enumerator **CONSTANT**

enumerator **LINEAR**

**brainflow.NoiseTypes**
*Values:*

enumerator **FIFTY**

enumerator **SIXTY**

enumerator **FIFTY_AND_SIXTY**

**brainflow.WaveletDenoisingTypes**
*Values:*

enumerator **VISUSHRINK**

enumerator **SURESHRINK**

**brainflow.ThresholdTypes**
*Values:*

enumerator **SOFT**

enumerator **HARD**

**brainflow.WaveletExtensionTypes**
*Values:*

enumerator **SYMMETRIC**

enumerator **PERIODIC**

**brainflow.NoiseEstimationLevelTypes**
*Values:*

enumerator **FIRST_LEVEL**

enumerator **ALL_LEVELS**

**brainflow.WaveletTypes**

*Values:*

enumerator **HAAR**

enumerator **DB1**

enumerator **DB2**

enumerator **DB3**

enumerator **DB4**

enumerator **DB5**

enumerator **DB6**

enumerator **DB7**

enumerator **DB8**

enumerator **DB9**

enumerator **DB10**

enumerator **DB11**

enumerator **DB12**

enumerator **DB13**

enumerator **DB14**

enumerator **DB15**

enumerator **BIOR1_1**

enumerator **BIOR1_3**

enumerator **BIOR1_5**

enumerator **BIOR2_2**

enumerator **BIOR2_4**

enumerator **BIOR2_6**

enumerator **BIOR2_8**

enumerator **BIOR3_1**

enumerator **BIOR3_3**

enumerator **BIOR3_5**

enumerator **BIOR3_7**

enumerator **BIOR3_9**

enumerator **BIOR4_4**

enumerator **BIOR5_5**

enumerator **BIOR6_8**

enumerator **COIF1**

enumerator **COIF2**

enumerator **COIF3**

enumerator **COIF4**

enumerator **COIF5**

enumerator **SYM2**

enumerator **SYM3**

enumerator **SYM4**

enumerator **SYM5**

enumerator **SYM6**

enumerator **SYM7**

enumerator **SYM8**

enumerator **SYM9**

enumerator **SYM10**

**brainflow.BrainFlowMetrics**

*Values:*

enumerator **MINDFULNESS**

enumerator **RESTFULNESS**

enumerator **USER_DEFINED**

**brainflow.BrainFlowClassifiers**

*Values:*

enumerator **DEFAULT_CLASSIFIER**

enumerator **DYN_LIB_CLASSIFIER**

enumerator **ONNX_CLASSIFIER**

**brainflow.brainflow.BoardDescr**

**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 config_board_with_bytes (byte[] bytes)**
send raw bytes to device, dont use it

**Parameters** **bytes** –

void **add_streamer**(string streamer_params, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
add streamer

**Parameters** **streamer_params** – supoprted formats file://filename:w file://filename:a streaming_board://ip:port

void **delete_streamer**(string streamer_params, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
delete streamer

**Parameters** **streamer_params** – supoprted formats file://filename:w file://filename:a streaming_board://ip:port

void **insert_marker**(double value, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
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>

**Returns** session status

**Returns** Master board id

int **get_board_id**()
summary> Get input params /summary>

**Returns** input params

int **get_board_data_count**(int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get number of packages in ringbuffer

**Parameters** **preset** – preset for device

**Returns** number of packages

**double[,] get_current_board_data (int num_samples,**
**int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**
get latest collected data, doesnt remove it from ringbuffer

**Parameters**

- **num_samples** –
- **preset** – preset for device

**Returns** latest collected data, can be less than "num_samples"

**double[,] get_current_board_data (double[,] result, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**
get latest collected data, doesnt remove it from ringbuffer

**Parameters**

- **preset** – preset for device
- **result** – array to store results, if provided BrainFlow does not allocate new memory

**Returns** latest collected data, can be less than "num_samples"

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

**Returns** collected data

**double[,] get_board_data (int num_datapoints)**
get collected data and remove it from ringbuffer

**Returns** collected data

**double[,] get_board_data (int num_datapoints, int preset)**
get collected data and remove it from ringbuffer

**Parameters**

- **num_datapoints** – number of datapoints to get
- **preset** – preset for device

**Returns** all collected data

**double[,] get_board_data (double[,] data_arr, int preset)**
get collected data and remove it from ringbuffer

**Parameters**

- **data_arr** – array to store results, if provided BrainFlow will not allocate memory
- **preset** – preset for device

**Returns** all collected data

### Public Members

int **board_id**
BrainFlow's board id

### Public Static Functions

void **release_all_sessions**()
release all sessions

int **get_sampling_rate**(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get sampling rate for this board id

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** sampling rate

int **get_package_num_channel**(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get row index in returned by *get_board_data()* 2d array which holds package nums

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** row num in 2d array

int **get_timestamp_channel**(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get row index which holds timestamps

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** row num in 2d array

int **get_marker_channel**(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get row index which holds marker

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** row num in 2d array

int **get_battery_channel**(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get row undex which holds battery level

**Parameters**

- **board_id** –

- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** row num in 2d array

int **get_num_rows**(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get number of rows in returned by *get_board_data()* 2d array

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** number of rows in 2d array

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

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of 10-20 locations

**int[] get_board_presets (int board_id)**
get presets for selected device

**Parameters** **board_id** –

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of strings

template<>
T **get_board_descr**<T>(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get board description

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If board id is not valid exit code is UNSUPPORTED_BOARD_ERROR

**Returns** board description

string **get_device_name**(int board_id, int preset = (int)BrainFlowPresets.DEFAULT_PRESET)
get device name

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** device name

string **get_version**()
get version

**Throws** **BrainFlowException** –

**Returns** version

**int[] get_eeg_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**
get row indices of EEG channels for this board, for some board we can not split EMG\EEG.. data and return the same array for all of them

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

**int[] get_emg_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**
get row indices of EMG channels for this board, for some board we can not split EMG\EEG.. data and return the same array for all of them

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

**int[] get_ecg_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**
get row indices of ECG channels for this board, for some board we can not split EMG\EEG.. data and return the same array for all of them

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws** **BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_eog_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

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

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_exg_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

get row indices of EXG channels for this board

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_eda_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

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

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_ppg_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

get row indeces which hold ppg data

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_accel_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

get row indices which hold accel data

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_rotation_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

get row indices which hold rotation data

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_analog_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

get row indices which hold analog data

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_gyro_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

get row indices which hold gyro data

**Parameters**

- `board_id` –
- `preset` – preset for device

**Throws** `BrainFlowException` – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

`int[] get_other_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)`

get other channels for this board

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

**int[] get_temperature_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**

get temperature channels for this board

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

**int[] get_resistance_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**

get resistance channels for this board

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

**int[] get_magnetometer_channels (int board_id, int preset=(int) BrainFlowPresets.DEFAULT_PRESET)**

get magnetometer channels for this board

**Parameters**

- **board_id** –
- **preset** – preset for device

**Throws BrainFlowException** – If this board has no such data exit code is UNSUPPORTED_BOARD_ERROR

**Returns** array of row nums

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** –

**brainflow.brainflow.BrainFlowError : public Exception**
BrainFlowError class to notify about errors

### Public Members

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

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

### Public Members

string **serial_port**
serial port name

string **mac_address**
MAC address

string **ip_address**
IP address

string **ip_address_aux**
IP address aux

string **ip_address_anc**
IP address anc

int **ip_port**
PORT

int **ip_port_aux**
PORT

int **ip_port_anc**
PORT

int **ip_protocol**
IP protocol, use IpProtocolTypes

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

string **file_aux**
file

string **file_anc**
file

int **master_board**
Master board, use BoardIds enums

**brainflow.brainflow.BrainFlowModelParams**
Describe model

### Public Members

int **metric**
metric to caclulate

int **classifier**
classifier to use

string **file**
path to model file

string **other_info**
other info

string **output_name**
output name

int **max_array_size**
max array size

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

### Public Static Functions

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

**Parameters** **log_level** –

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

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

**Parameters**

- **log_level** –
- **message** –

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, int filter_type, double ripple)**
perform lowpass filter, unlike other bindings instead in-place calculation it returns new array

**Parameters**

- **data** –
- **sampling_rate** –
- **cutoff** –
- **order** –
- **filter_type** –
- **ripple** –

**Returns** filtered data

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

**Parameters**

- **data** –
- **sampling_rate** –

- **noise_type** –

**Returns** filtered data

**double[] perform_highpass (double[] data, int sampling_rate, double cutoff, int order, int filter_type, double ripple)**

perform highpass filter, unlike other bindings instead in-place calculation it returns new array

**Parameters**

- **data** –
- **sampling_rate** –
- **cutoff** –
- **order** –
- **filter_type** –
- **ripple** –

**Returns** filtered data

**double[] perform_bandpass (double[] data, int sampling_rate, double start_freq, double stop_freq, int order, int filter_type, double ripple)**

perform bandpass filter, unlike other bindings instead in-place calculation it returns new array

**Parameters**

- **data** –
- **sampling_rate** –
- **start_freq** –
- **stop_freq** –
- **order** –
- **filter_type** –
- **ripple** –

**Returns** filtered data

**double[] perform_bandstop (double[] data, int sampling_rate, double start_freq, double stop_freq, int order, int filter_type, double ripple)**

perform bandstop filter, unlike other bindings instead in-place calculation it returns new array

**Parameters**

- **data** –
- **sampling_rate** –
- **start_freq** –
- **stop_freq** –
- **order** –
- **filter_type** –
- **ripple** –

**Returns** filtered data

`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

**Parameters**

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

**Returns** filered data

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

**Parameters**

- `data` –
- `operation` –

**Returns** data with removed trend

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

**Parameters**

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

**Returns** data after downsampling

`double[] detect_peaks_z_score (double[] data, int lag, double threshold, double influence)`
detect peaks using z score algorithm

`double calc_stddev (double[] data, int start_pos, int end_pos)`
calc stddev

///

**Parameters**

- `data` –
- `start_pos` –
- `end_pos` –

**Returns** stddev

`double get_railed_percentage (double[] data, int gain)`
get railed percentage

**Parameters**

- **data** –
- **gain** –

**Returns** railed

**double get_oxygen_level (double[] ppg_ir, double[] ppg_red, int sampling_rate, double coef1=1.5958422, double coef2=-34.6596622, double coef3=112.6898759)**

calc oxygen level from ppg values

///

**Parameters**

- **coef1** – appxorimation coef for power of 2
- **coef2** – approximation coef
- **coef3** – intercept for approximation

**Returns** stddev

**double get_heart_rate (double[] ppg_ir, double[] ppg_red, int sampling_rate, int fft_size)**

calc heart rate

**Parameters** **fft_size** – recommended 8192

**Returns** stddev

**Tuple< double[], int[]> perform_wavelet_transform (double[] data, int wavelet, int decomposition_level, int extension)**

perform wavelet transform

**Parameters**

- **data** – data for wavelet transform
- **wavelet** – use WaveletTypes enum
- **extension** – use WaveletExtensionTypes enum
- **decomposition_level** – decomposition level

**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

**double[] perform_inverse_wavelet_transform (Tuple< double[], int[]> wavelet_data, int original_data_len, int wavelet, int decomposition_level, int extension)**

perform inverse wavelet transorm

**Parameters**

- **wavelet_data** – tuple returned by perform_wavelet_transform
- **original_data_len** – size of original data before direct wavelet transform
- **wavelet** – use WaveletTypes enum
- **decomposition_level** – level of decomposition
- **extension** – use WaveletExtensionTypes enum

**Returns** restored data

`double[] perform_wavelet_denoising (double[] data, int wavelet, int decomposition_level, int wavelet_denoising=(int) WaveletDenoisingTypes.SURESHRINK, int threshold=(int) ThresholdTypes.HARD, int extenstion_type=(int) WaveletExtensionTypes.SYMMETRIC, int noise_level=(int) NoiseEstimationLevelTypes.FIRST_LEVEL)`
perform wavelet based denoising

**Parameters**

- **data** – data for denoising
- **wavelet** – use WaveletTypes enum
- **decomposition_level** – level of decomposition in wavelet transform
- **extenstion_type** – use WaveletExtensionTypes enum
- **noise_level** – use NoiseEstimationLevelTypes enum
- **threshold** – use ThresholdTypes enum
- **wavelet_denoising** – use WaveletDenoisingTypes enum

**Returns** denoised data

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

**Parameters**

- **data** – data for csp
- **labels** – labels for each class

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

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

**Parameters**

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

**Returns** array of the size specified in window_len

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

**Parameters**

- **data** – data for fft
- **start_pos** – start pos
- **end_pos** – end pos, end_pos - start_pos must be even
- **window** – window function

**Returns** complex array of size N / 2 + 1 of fft data

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

**Parameters** **data** – data from perform_fft

**Returns** restored data

**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

**Parameters** **file_name** –

**Returns**

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

**Parameters** **value** –

**Returns** nearest power of two

**Tuple< double[], double[]> get_custom_band_powers (double[,] data, Tuple< double, double >[] bands, int[] channels, int sampling_rate, bool apply_filters)**
calculate avg and stddev bandpowers across channels

**Parameters**

- **data** – 2d array with values
- **bands** – bands to calculate
- **channels** – rows of data array which should be used for calculation
- **sampling_rate** – sampling rate
- **apply_filters** – apply bandpass and bandstop filters before calculation

**Returns** Tuple of avgs and stddev arrays

**Tuple< double[,], double[,], double[,], double[,]> perform_ica (double[,] data, int num_components)**
Calculate ICA

**Parameters**

- **data** –
- **num_components** –

**Returns**

`Tuple< double[,], double[,], double[,], double[,]> perform_ica (double[,] data, int num_components, int[] channels)`
Calculate ICA

**Parameters**

- `data` –
- `num_components` –
- `channels` –

**Returns**

`Tuple< double[], double[]> get_avg_band_powers (double[,] data, int[] channels, int sampling_rate, bool apply_filters)`
calculate avg and stddev bandpowers across channels

**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

**Returns** Tuple of avgs and stddev arrays

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

**Parameters**

- `data` – data for PSD
- `start_pos` – start pos
- `end_pos` – end pos, end_pos - start_pos must be even
- `sampling_rate` – sampling rate
- `window` – window function

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

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

**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

**Returns** Tuple of ampls and freqs arrays

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

**Returns** band power

string **get_version**()
get version

**Throws** **BrainFlowException** –

**Returns** version

**unsafe void perform_lowpass (double[,] data, int row_num, int sampling_rate, double cutoff, int order, int filter_type, double ripple)**
perform lowpass filter for specified row number in-place

**Parameters**

- **data** –
- **row_num** –
- **sampling_rate** –
- **cutoff** –
- **order** –
- **filter_type** –
- **ripple** –

**unsafe void remove_environmental_noise (double[,] data, int row_num, int sampling_rate, int noise_type)**
remove env noise using notch filter in-place

**Parameters**

- **data** –
- **row_num** –
- **sampling_rate** –
- **noise_type** –

**unsafe void perform_highpass (double[,] data, int row_num, int sampling_rate, double cutoff, int order, int filter_type, double ripple)**
perform highpass filter in-place

**Parameters**

- **data** –
- **sampling_rate** –
- **cutoff** –
- **order** –

- filter_type –
- ripple –

unsafe void perform_bandpass (double[,] data, int row_num, int sampling_rate, double start_freq, double stop_freq, int order, int filter_type, double ripple)

perform bandpass filter in-place

**Parameters**

- data –
- row_num –
- sampling_rate –
- start_freq –
- stop_freq –
- order –
- filter_type –
- ripple –

unsafe void perform_bandstop (double[,] data, int row_num, int sampling_rate, double start_freq, double stop_freq, int order, int filter_type, double ripple)

perform bandstop filter in-place

**Parameters**

- data –
- row_num –
- sampling_rate –
- start_freq –
- stop_freq –
- order –
- filter_type –
- ripple –

**Returns** filtered data

unsafe void perform_rolling_filter (double[,] data, int row_num, int period, int operation)

perform moving average or moving median filter, unlike other bindings instead in-place calculation it returns new array

**Parameters**

- data –
- row_num –
- period –
- operation –

**Returns** filered data

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

**Parameters**

- data –
- row_num –
- operation –

**Returns** data with removed trend

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

**Parameters**

- data –
- row_num –
- period –
- operation –

**Returns** data after downsampling

unsafe double[] detect_peaks_z_score (double[,] data, int row_num, int lag, double threshold, double influence)
detect peaks using z score algorithm

unsafe double calc_stddev (double[,] data, int row_num, int start_pos, int end_pos)
calc stddev

///

**Parameters**

- data –
- row_num –
- start_pos –
- end_pos –

**Returns** stddev

unsafe double get_railed_percentage (double[,] data, int row_num, int gain)
get railed percentage

**Parameters**

- data –
- row_num –
- gain –

**Returns** railed

`unsafe Tuple< double[], int[]> perform_wavelet_transform (double[,] data, int row_num, int wavelet, int decomposition_level, int extension)`
perform wavelet transform

**Parameters**

- **data** – data for wavelet transform
- **row_num** –
- **wavelet** – use WaveletTypes enum
- **extension** – use WaveletExtensionTypes enum
- **decomposition_level** – decomposition level

**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

`unsafe void perform_wavelet_denoising (double[,] data, int row_num, int wavelet, int decomposition_level, int wavelet_denoising=(int) WaveletDenoisingTypes.SURESHRINK, int threshold=(int) ThresholdTypes.HARD, int extenstion_type=(int) WaveletExtensionTypes.SYMMETRIC, int noise_level=(int) NoiseEstimationLevelTypes.FIRST_LEVEL)`
perform wavelet based denoising

**Parameters**

- **data** – data for denoising
- **row_num** –
- **wavelet** – use WaveletTypes enum
- **decomposition_level** – level of decomposition in wavelet transform
- **extenstion_type** – use WaveletExtensionTypes enum
- **noise_level** – use NoiseEstimationLevelTypes enum
- **threshold** – use ThresholdTypes enum
- **wavelet_denoising** – use WaveletDenoisingTypes enum

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

**Parameters**

- **data** – data for fft
- **row_num** –
- **start_pos** – start pos
- **end_pos** – end pos, end_pos - start_pos must be even
- **window** – window function

**Returns** complex array of size N / 2 + 1 of fft data

`unsafe Tuple< double[], double[]> get_psd (double[,] data, int row_num, int start_pos, int end_pos, int sampling_rate, int window)`
calculate PSD

**Parameters**

- **data** – data for PSD
- **row_num** –
- **start_pos** – start pos
- **end_pos** – end pos, end_pos - start_pos must be even
- **sampling_rate** – sampling rate
- **window** – window function

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

**unsafe Tuple< double[], double[]> get_psd_welch (double[,] data, int row_num, int nfft, int overlap, int sampling_rate, int window)**
calculate PSD using Welch method

**Parameters**

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

**Returns** Tuple of ampls and freqs arrays

## 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 **release_all**()
release all classifiers

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

**Parameters** **log_level** –

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

**Parameters**

- **log_level** –
- **message** –

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** –

string **get_version**()
get version

**Throws** **BrainFlowException** –

**Returns** version

**brainflow.brainflow.PlatformHelper**

## 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
```

(continues on next page)

(continued from previous page)

```
sys <- NULL
type_map <- NULL

.onLoad <- function(libname, pkgname)
{
    brainflow_python <<- import('brainflow', delay_load = TRUE)
    np <<- import('numpy', delay_load = TRUE, convert = FALSE)
    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.**MLModel**(*params*)

MLModel for inference

**Method Summary**

**static release_all**()
release all sessions

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

**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

**static get_version()**
get version

**prepare()**
prepare model

**release()**
release model

**predict**(*input_data*)
perform inference for input data

**class** brainflow.**NoiseTypes**
Bases: int32

Store noise types

**class** brainflow.**FilterTypes**
Bases: int32

Store all possible filters

**class** brainflow.**DetrendOperations**
Bases: int32

Store possible detrend operations

**class** brainflow.**BoardIds**
Bases: int32

Store all supported board ids

**class** brainflow.**IpProtocolTypes**
Bases: int32

Store all possible IP protocols

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

**Method Summary**

**static get_version()**
get version

**static log_message**(*log_level*, *message*)
write message to Data Filter logger

**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*, *start_freq*, *stop_freq*, *order*, *filter_type*, *ripple*)
perform bandpass filtering

**static perform_bandstop**(*data*, *sampling_rate*, *start_freq*, *stop_freq*, *order*, *filter_type*, *ripple*)
perform bandpass filtering

**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 restore_data_from_wavelet_detailed_coeffs**(*data*, *wavelet*, *decomposition_level*, *level_to_restore*)
restore data from a single wavelet level

**static detect_peaks_z_score**(*data*, *lag*, *threshold*, *influence*)
peaks detection using z score algorithm

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

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

**static perform_wavelet_denoising**(*data*, *wavelet*, *decomposition_level*, *denoising*, *threshold*, *extention*, *noise_level*)
perform wavelet denoising

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

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

**static calc_stddev**(*data*)
calc stddev

**static get_railed_percentage**(*data*, *gain*)
calc railed percentage

**static get_oxygen_level**(*ppg_ir*, *ppg_red*, *sampling_rate*, *coef1*, *coef2*, *coef3*)
calc oxygen level

**static get_heart_rate**(*ppg_ir*, *ppg_red*, *sampling_rate*, *fft_size*)
calc heart rate

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

**static perform_ifft**(*fft_data*)
perform inverse fft

**static get_custom_band_powers**(*data*, *start_freqs*, *stop_freqs*, *channels*, *sampling_rate*, *apply_filters*)
calculate average band powers

**static perform_ica_select_channels**(*data*, *num_components*, *channels*)
calculate ica

**static perform_ica**(*data*, *num_components*)
calculate ica

**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.**WaveletTypes**
Bases: int32

Store all possible Wavelet Types

**class** brainflow.**BrainFlowExitCodes**
Bases: int32

Store all possible exit codes

**class** brainflow.**ThresholdTypes**
Bases: int32

Store all possible threshold types

**class** brainflow.**WaveletExtensionTypes**
Bases: int32

Store all possible extensions

**class** brainflow.**WindowOperations**
Bases: int32

Store window functions

**class** brainflow.**NoiseEstimationLevelTypes**
Bases: int32

Store all possible noise estimation levels

class brainflow.**BrainFlowClassifiers**
Bases: int32

Store supported classifiers

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

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

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

BoardShim constructor

**Method Summary**

**static release_all_sessions**()
release all sessions

**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*, *preset*)
get sampling rate for provided board id

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

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

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

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

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

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

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

**static get_board_presets**(*board_id*)
get supported presets

**static get_version**()
get version

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

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

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

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

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

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

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

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

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

**static get_rotation_channels**(*board_id*, *preset*)
get rotation channels for provided board id

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

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

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

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

**static get_magnetometer_channels**(*board_id*, *preset*)
get magnetometer channels for provided board id

**prepare_session**()
prepare BoardShim session

**config_board**(*config*)
send string to the board

**config_board_with_bytes**(*bytes*)
send bytes to the board, do not use it

**add_streamer**(*streamer*, *preset*)
add streamer

**delete_streamer**(*streamer*, *preset*)
delete streamer

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

**stop_stream**()
stop acqusition

**release_session**()
release session

**insert_marker**(*value*, *preset*)
insert marker

**get_board_data_count**(*preset*)
get amount of datapoints in internal buffer

**get_board_data**(*num_datapoints*, *preset*)
get required amount of data by specifying it and remove it from the buffer

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

**is_prepared**()
check if brainflow session prepared

**class** brainflow.**AggOperations**
Bases: int32

Store all supported Agg Operations

**class** brainflow.**LogLevels**
Bases: int32

Store all possible log levels

**class** brainflow.**BrainFlowPresets**
Bases: int32

Store all supported presets

**class** brainflow.**WaveletDenoisingTypes**
Bases: int32

Store all possible denoising methods for wavelets

**class** brainflow.**BrainFlowMetrics**
Bases: int32

Store all supported metrics

## 3.7 Julia API Reference

Julia binding calls C/C++ 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.

Example:

```
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.add_streamer("file://data_default.csv:w", 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)
```

## 3.8 Rust API Reference

Rust binding calls C/C++ code as any other binding. Use Rust examples and API reference for other languaes as a starting point.

Example:

```
use std::{thread, time::Duration};

use brainflow::{board_shim, brainflow_input_params::BrainFlowInputParamsBuilder,␣
↪BoardIds, BrainFlowPresets,};

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let params = BrainFlowInputParamsBuilder::default().build();
    let board = board_shim::BoardShim::new(BoardIds::SyntheticBoard, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let data = board.get_board_data(Some(10), BrainFlowPresets::DefaultPreset).unwrap();
    board.release_session().unwrap();

    println!("{}", data.len());
    println!("{:?}", data);
```

(continues on next page)

(continued from previous page)

```
}
```

## 3.9 Typescript API Reference

Typescript binding calls C/C++ code as any other binding. Use Typescript examples and API reference for other languaes as a starting point.

Example:

```
import {BoardIds, BoardShim} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const board = new BoardShim (BoardIds.SYNTHETIC_BOARD, {});
    board.prepareSession();
    board.startStream();
    await sleep (3000);
    board.stopStream();
    const data = board.getBoardData();
    board.releaseSession();
    console.info('Data');
    console.info(data);
}

runExample ();
```

CHAPTER

FOUR

# DATA FORMAT DESCRIPTION

## 4.1 Units of Measure

For EXG channels BrainFlow returns uV wherever possible.

For timestamps BrainFlow uses UNIX timestamp, this counter 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 packages were received.

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

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

## 4.2 BrainFlow Presets

Each board can accumulate data inside one of three predefined buffers. In BrainFlow they are called presets, possible values are:

- *BrainFlowPresets.DEFAULT_PRESET*
- *BrainFlowPresets.AUXILIARY_PRESET*
- *BrainFlowPresets.ANCILLARY_PRESET*

Each of them has it's own sampling rate, timestamp and package id. Other data types depend on exact device.

For almost all devices only *BrainFlowPresets.DEFAULT_PRESET* is available. But for some devices, especially if they stream different types of data with different sampling rates(e.g. Muse) we store these data types in different presets. All methods like:

```
insert_marker
get_board_data
get_current_board_data
add_streamer
get_sampling_rate
get_timestamp_channel
get_eeg_channels
# etc
```

Have an optional preset parameter with default value *BrainFlowPresets.DEFAULT_PRESET* if programming language supports default values for function arguments. Here is a code sample that you can use as a referece:

```
using BrainFlow

BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)

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

BrainFlow.prepare_session(board_shim)
BrainFlow.config_board("p50", board_shim) # to enable ppg use p61, p50 enables aux(5th␣
↪eeg) channel and smth else
BrainFlow.add_streamer("file://default_from_streamer.csv:w", board_shim, BrainFlow.
↪DEFAULT_PRESET)
BrainFlow.add_streamer("file://aux_from_streamer.csv:w", board_shim, BrainFlow.AUXILIARY_
↪PRESET)
BrainFlow.add_streamer("file://anc_from_streamer.csv:w", board_shim, BrainFlow.ANCILLARY_
↪PRESET)
BrainFlow.start_stream(board_shim)
sleep(10)
BrainFlow.stop_stream(board_shim)
data_default = BrainFlow.get_board_data(board_shim, BrainFlow.DEFAULT_PRESET) # contains␣
↪eeg data
data_aux = BrainFlow.get_board_data(board_shim, BrainFlow.AUXILIARY_PRESET) # contains␣
↪accel and gyro data
data_anc = BrainFlow.get_board_data(board_shim, BrainFlow.ANCILLARY_PRESET) # contains␣
↪ppg data
BrainFlow.release_session(board_shim)

BrainFlow.write_file(data_default, "default.csv", "w")
BrainFlow.write_file(data_aux, "aux.csv", "w")
BrainFlow.write_file(data_anc, "anc.csv", "w")
```

You can get all presets availabe for your device using `BoardShim.get_board_presets(board_id)` method.

## 4.3 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)
```

(continues on next page)

(continued from previous page)

```
# 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 be different.**

**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 dummy BrainFlow boards and it will help you during developemnent and testing.

### 4.3.1 Getting All Info About Device And Supported Channels

There is a method `get_board_descr(int board_id)`. You can use it to get all info about specified device and BrainFlow channels for this board.

```
from pprint import pprint

import brainflow
from brainflow.board_shim import BoardShim, BoardIds

board_id = BoardIds.SYNTHETIC_BOARD.value
pprint(BoardShim.get_board_descr(board_id))
```

```
{"accel_channels": [17, 18, 19],
 "battery_channel": 29,
 "ecg_channels": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
 "eda_channels": [23],
 "eeg_channels": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
 "eeg_names": "Fz,C3,Cz,C4,Pz,PO7,Oz,PO8,F5,F7,F3,F1,F2,F4,F6,F8",
 "emg_channels": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
 "eog_channels": [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16],
 "gyro_channels": [20, 21, 22],
 "marker_channel": 31,
 "name": "Synthetic",
 "num_rows": 32,
 "package_num_channel": 0,
 "ppg_channels": [24, 25],
 "resistance_channels": [27, 28],
 "sampling_rate": 250,
 "temperature_channels": [26],
 "timestamp_channel": 30}
```

## 4.4 Other Channels

Some boards have pretty unique data types and we do not have dedicated methods for them, for such devices we return data in `get_other_channels()`. Please refer to the source code to get more info about it.

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

from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds,␣
↪BrainFlowPresets


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)
```

(continues on next page)

(continued from previous page)

```
    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('--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='')
    parser.add_argument('--master-board', type=int, help='master board id for streaming␣
↪and playback boards',
                        required=False, default=BoardIds.NO_BOARD)
    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
    params.master_board = args.master_board

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

from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds,␣
↪BrainFlowPresets


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('--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='')
    parser.add_argument('--master-board', type=int, help='master board id for streaming␣
↪and playback boards',
                        required=False, default=BoardIds.NO_BOARD)
    parser.add_argument('--preset', type=int, help='preset for streaming and playback␣
↪boards',
                        required=False, default=BrainFlowPresets.DEFAULT_PRESET)
    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
```

(continues on next page)

(continued from previous page)

```
    params.file = args.file
    params.master_board = args.master_board
    params.preset = args.preset

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

    board.start_stream()
    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 time

import numpy as np
import pandas as pd
from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter


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')
```

(continues on next page)

(continued from previous page)

```
    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))


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

### 5.1.4 Python Downsample Data

```
import time

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, 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(20)
    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)
```

(continues on next page)

(continued from previous page)

```
        print('Downsampled data for channel %d:' % channel)
        print(downsampled_data)


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

### 5.1.5 Python Transforms

```
import time

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, WindowOperations, WaveletTypes


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(256)
    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],␣
↪WaveletTypes.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],
```

(continues on next page)

(continued from previous page)

```
                                                                    WaveletTypes.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], WindowOperations.NO_WINDOW.
↪value)
        # len of fft_data is N / 2 + 1
        restored_fft_data = DataFilter.perform_ifft(fft_data)
        print('Restored data after fft for channel %d:' % channel)
        print(restored_fft_data)


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

### 5.1.6 Python Signal Filtering

```
import time

import matplotlib
import numpy as np
import pandas as pd

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))
```

(continues on next page)

(continued from previous page)

```
    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), 2.0, 50.0, 4,
                                        FilterTypes.BESSEL_ZERO_PHASE, 0)
        elif count == 1:
            DataFilter.perform_bandstop(data[channel], BoardShim.get_sampling_rate(board_
↪id), 48.0, 52.0, 3,
                                        FilterTypes.BUTTERWORTH_ZERO_PHASE, 0)
        elif count == 2:
            DataFilter.perform_lowpass(data[channel], BoardShim.get_sampling_rate(board_
↪id), 50.0, 5,
                                       FilterTypes.CHEBYSHEV_TYPE_1_ZERO_PHASE, 1)
        elif count == 3:
            DataFilter.perform_highpass(data[channel], BoardShim.get_sampling_rate(board_
↪id), 2.0, 4,
                                        FilterTypes.BUTTERWORTH, 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 time

import matplotlib
import numpy as np
import pandas as pd

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

(continues on next page)

(continued from previous page)

```
from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, AggOperations, WaveletTypes,␣
↪NoiseEstimationLevelTypes, \
    WaveletExtensionTypes, ThresholdTypes, WaveletDenoisingTypes


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_current_board_data(500)
    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 parameters
        else:
            DataFilter.perform_wavelet_denoising(data[channel], WaveletTypes.BIOR3_9, 3,
                                                 WaveletDenoisingTypes.SURESHRINK,␣
↪ThresholdTypes.HARD,
                                                 WaveletExtensionTypes.SYMMETRIC,␣
↪NoiseEstimationLevelTypes.FIRST_LEVEL)

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

(continues on next page)

(continued from previous page)

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

### 5.1.8 Python MNE Integration

```
import time

import matplotlib

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

from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds

import mne


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 time

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels, BoardIds
from brainflow.data_filter import DataFilter, WindowOperations, DetrendOperations


def main():
    BoardShim.enable_dev_board_logger()

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


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

### 5.1.10 Python EEG Metrics

```
import argparse
import time

from brainflow.board_shim import BoardShim, BrainFlowInputParams, LogLevels
from brainflow.data_filter import DataFilter
from brainflow.ml_model import MLModel, BrainFlowMetrics, BrainFlowClassifiers,␣
↪BrainFlowModelParams


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

(continues on next page)

(continued from previous page)

```
    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 = bands[0]
    print(feature_vector)

    mindfulness_params = BrainFlowModelParams(BrainFlowMetrics.MINDFULNESS.value,
                                              BrainFlowClassifiers.DEFAULT_CLASSIFIER.
↪value)
    mindfulness = MLModel(mindfulness_params)
    mindfulness.prepare()
    print('Mindfulness: %s' % str(mindfulness.predict(feature_vector)))
    mindfulness.release()

    restfulness_params = BrainFlowModelParams(BrainFlowMetrics.RESTFULNESS.value,
                                              BrainFlowClassifiers.DEFAULT_CLASSIFIER.
↪value)
    restfulness = MLModel(restfulness_params)
    restfulness.prepare()
    print('Restfulness: %s' % str(restfulness.predict(feature_vector)))
    restfulness.release()


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

### 5.1.11 Python ICA

```
import time
import numpy as np

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

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

(continues on next page)

(continued from previous page)

```
from brainflow.data_filter import DataFilter


def main():
    board_id = BoardIds.SYNTHETIC_BOARD
    eeg_channels = BoardShim.get_eeg_channels(board_id)

    params = BrainFlowInputParams()
    board = BoardShim(board_id, params)
    board.prepare_session()
    board.start_stream()
    time.sleep(10)
    data = board.get_board_data(500)
    board.stop_stream()
    board.release_session()

    channel_to_use = eeg_channels[4]
    data = data[channel_to_use, :]
    # provide 5 chunks of data for components selection
    data = data.reshape(5, 100)
    data = np.ascontiguousarray(data)
    w, k, a, s = DataFilter.perform_ica(data, 2)
    fig, axs = plt.subplots(2, 1)
    axs[0].plot(s[0, :])
    axs[0].set_title('Unmixed signal 1')
    axs[1].plot(s[1, :])
    axs[1].set_title('Unmixed signal 2')
    plt.savefig('unmixed_signal.png')


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

### 5.1.12 Python Real Time Plot

Extra requirements for this code sample:

```
PyQt5==5.15.7
PyQt5-Qt5==5.15.2
PyQt5-sip==12.11.0
pyqtgraph==0.12.4
brainflow
```

```
import argparse
import logging

import pyqtgraph as pg
from brainflow.board_shim import BoardShim, BrainFlowInputParams, BoardIds
from brainflow.data_filter import DataFilter, FilterTypes, DetrendOperations
from pyqtgraph.Qt import QtGui, QtCore
```

(continues on next page)

(continued from previous page)

```
class Graph:
    def __init__(self, board_shim):
        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_timeseries()

        timer = QtCore.QTimer()
        timer.timeout.connect(self.update)
        timer.start(self.update_speed_ms)
        QtGui.QApplication.instance().exec_()

    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)
            curve = p.plot()
            self.curves.append(curve)

    def update(self):
        data = self.board_shim.get_current_board_data(self.num_points)
        for count, channel in enumerate(self.exg_channels):
            # plot timeseries
            DataFilter.detrend(data[channel], DetrendOperations.CONSTANT.value)
            DataFilter.perform_bandpass(data[channel], self.sampling_rate, 3.0, 45.0, 2,
                                        FilterTypes.BUTTERWORTH_ZERO_PHASE, 0)
            DataFilter.perform_bandstop(data[channel], self.sampling_rate, 48.0, 52.0, 2,
                                        FilterTypes.BUTTERWORTH_ZERO_PHASE, 0)
            DataFilter.perform_bandstop(data[channel], self.sampling_rate, 58.0, 62.0, 2,
                                        FilterTypes.BUTTERWORTH_ZERO_PHASE, 0)
            self.curves[count].setData(data[channel].tolist())

        self.app.processEvents()
```

(continues on next page)

(continued from previous page)

```
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
    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='')
    parser.add_argument('--master-board', type=int, help='master board id for streaming␣
↪and playback boards',
                        required=False, default=BoardIds.NO_BOARD)
    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
    params.master_board = args.master_board

    board_shim = BoardShim(args.board_id, params)
    try:
        board_shim.prepare_session()
        board_shim.start_stream(450000, args.streamer_params)
```

(continues on next page)

(continued from previous page)

```
        Graph(board_shim)
    except BaseException:
        logging.warning('Exception', exc_info=True)
    finally:
        logging.info('End')
        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++)
        {
```

(continues on next page)

(continued from previous page)

```
            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 ("--ip-address-aux"))
            {
                params.ip_address_aux = args[i + 1];
            }
            if (args[i].equals ("--ip-address-anc"))
            {
                params.ip_address_anc = 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-port-aux"))
            {
                params.ip_port_aux = Integer.parseInt (args[i + 1]);
            }
            if (args[i].equals ("--ip-port-anc"))
            {
                params.ip_port_anc = 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"))
            {
```

(continues on next page)

(continued from previous page)

```
                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];
            }
            if (args[i].equals ("--file-aux"))
            {
                params.file_aux = args[i + 1];
            }
            if (args[i].equals ("--file-anc"))
            {
                params.file_anc = args[i + 1];
            }
            if (args[i].equals ("--master-board"))
            {
                params.master_board = Integer.parseInt (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 ();
        BoardIds board_id = BoardIds.SYNTHETIC_BOARD;
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO, "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 ();
        BoardIds board_id = BoardIds.SYNTHETIC_BOARD;

        BoardShim board_shim = new BoardShim (board_id, params);
        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO, "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_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);
            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.WaveletExtensionTypes;
import brainflow.WaveletTypes;
import brainflow.WindowOperations;

public class Transforms
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        BoardIds board_id = BoardIds.SYNTHETIC_BOARD;
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO, "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
```

(continues on next page)

(continued from previous page)

```
            // 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]],
                    WaveletTypes.DB4, 3, WaveletExtensionTypes.SYMMETRIC);
            // print approximation coeffs
            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, WaveletTypes.DB4, 3,␣
↪WaveletExtensionTypes.SYMMETRIC);
            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,␣
↪WindowOperations.NO_WINDOW);
            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
```

(continues on next page)

(continued from previous page)

```
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        BoardIds board_id = BoardIds.SYNTHETIC_BOARD;
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO, "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 ();

        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), 50.0, 4,
                            FilterTypes.BESSEL, 0.0);
                    break;
                case 1:
                    DataFilter.perform_highpass (data[eeg_channels[i]], BoardShim.get_
↪sampling_rate (board_id), 3.0, 4,
                            FilterTypes.BUTTERWORTH, 0.0);
                    break;
                case 2:
                    DataFilter.perform_bandpass (data[eeg_channels[i]], BoardShim.get_
↪sampling_rate (board_id), 3.0,
                            50.0, 4, FilterTypes.CHEBYSHEV_TYPE_1, 1.0);
                    break;
                case 3:
                    DataFilter.perform_bandstop (data[eeg_channels[i]], BoardShim.get_
↪sampling_rate (board_id), 48.0,
                            52.0, 4, FilterTypes.CHEBYSHEV_TYPE_1, 1.0);
                    break;
                default:
                    DataFilter.remove_environmental_noise (data[eeg_channels[i]],
                            BoardShim.get_sampling_rate (board_id), NoiseTypes.FIFTY);;
                    break;
            }
        }
```

(continues on next page)

(continued from previous page)

```
        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 java.util.Arrays;

import brainflow.AggOperations;
import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
import brainflow.LogLevels;
import brainflow.NoiseEstimationLevelTypes;
import brainflow.ThresholdTypes;
import brainflow.WaveletDenoisingTypes;
import brainflow.WaveletExtensionTypes;
import brainflow.WaveletTypes;

public class Denoising
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        BoardIds board_id = BoardIds.SYNTHETIC_BOARD;
        BoardShim board_shim = new BoardShim (board_id, params);

        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        BoardShim.log_message (LogLevels.LEVEL_INFO, "Start sleeping in the main thread
↪");
        Thread.sleep (7000);
        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 (512);
        for (int i = 0; i < num_rows; i++)
        {
            System.out.println (Arrays.toString (data[i]));
        }
```

(continues on next page)

(continued from previous page)

```
        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);
                    break;
                case 1:
                    DataFilter.perform_rolling_filter (data[eeg_channels[i]], 3,␣
↪AggOperations.MEDIAN);
                    break;
                // if methods above dont work good for you you should try wavelet based
                // denoising
                default:
                    // try different functions and different decomposition levels here
                    DataFilter.perform_wavelet_denoising (data[eeg_channels[i]],␣
↪WaveletTypes.BIOR3_9, 3,
                            WaveletDenoisingTypes.SURESHRINK, ThresholdTypes.HARD,␣
↪WaveletExtensionTypes.SYMMETRIC,
                            NoiseEstimationLevelTypes.FIRST_LEVEL);
                    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.List;

import org.apache.commons.lang3.tuple.Pair;

import brainflow.BoardDescr;
import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;
```

(continues on next page)

(continued from previous page)

```
import brainflow.DetrendOperations;
import brainflow.LogLevels;
import brainflow.WindowOperations;

public class BandPower
{

    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        BoardIds board_id = BoardIds.SYNTHETIC_BOARD;
        BoardShim board_shim = new BoardShim (board_id, params);
        BoardDescr board_descr = BoardShim.get_board_descr (BoardDescr.class, board_id);
        int sampling_rate = board_descr.sampling_rate;
        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, "Start sleeping in the main thread
↪");
        Thread.sleep (10000);
        board_shim.stop_stream ();
        double[][] data = board_shim.get_board_data ();
        board_shim.release_session ();

        List<Integer> eeg_channels = board_descr.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);
        Pair<double[], double[]> psd = DataFilter.get_psd_welch (data[eeg_channel], nfft,
↪ nfft / 2, sampling_rate,
                WindowOperations.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);
        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.tuple.Pair;

import brainflow.BoardShim;
import brainflow.BrainFlowClassifiers;
import brainflow.BrainFlowInputParams;
```

(continues on next page)

(continued from previous page)

```
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, "Start sleeping in the main thread
↪");
        // recommended window size for eeg metric calculation is at least 4 seconds,
        // bigger is better
        Thread.sleep (5000);
        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 = bands.getLeft ();
        BrainFlowModelParams model_params = new BrainFlowModelParams (BrainFlowMetrics.
↪MINDFULNESS,
                BrainFlowClassifiers.DEFAULT_CLASSIFIER);
        MLModel model = new MLModel (model_params);
        model.prepare ();
        System.out.print ("Score: " + model.predict (feature_vector)[0]);
        model.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"))
```

(continues on next page)

(continued from previous page)

```
            {
                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;
    }
}
```

### 5.2.10 Java ICA

```
package brainflow.examples;

import java.util.List;

import brainflow.BoardIds;
import brainflow.BoardShim;
import brainflow.BrainFlowInputParams;
import brainflow.DataFilter;

public class ICA
{

```

(continues on next page)

(continued from previous page)

```
    public static void main (String[] args) throws Exception
    {
        // use synthetic board for demo
        BoardShim.enable_board_logger ();
        BrainFlowInputParams params = new BrainFlowInputParams ();
        BoardIds board_id = BoardIds.SYNTHETIC_BOARD;
        BoardShim board_shim = new BoardShim (board_id, params);
        board_shim.prepare_session ();
        board_shim.start_stream (3600);
        Thread.sleep (10000);
        board_shim.stop_stream ();
        double[][] data = board_shim.get_board_data (500);
        board_shim.release_session ();

        int[] eeg_channels = BoardShim.get_eeg_channels (board_id);
        int eeg_channel = eeg_channels[1];
        double[][] ica_data = DataFilter.reshape_data_to_2d (5, 100, data[eeg_channel]);
        List<double[][]> ica = DataFilter.perform_ica (ica_data, 2);
        System.out.println ("Completed");
    }
}
```

## 5.3 C#

### 5.3.1 C# Read Data from a Board

```
using System;

using brainflow;
using brainflow.math;


namespace examples
{
    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 ();
            System.Threading.Thread.Sleep (5000);
            board_shim.stop_stream ();
            double[,] unprocessed_data = board_shim.get_current_board_data (20);
```

(continues on next page)

(continued from previous page)

```
            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];
                }
                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"))
```

(continues on next page)

(continued from previous page)

```
                {
                    input_params.file = args[i + 1];
                }
            }
            return board_id;
        }
    }
}
```

### 5.3.2 C# Markers

```
using System;

using brainflow;
using brainflow.math;


namespace examples
{
    class Markers
    {
        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 ();
            board_shim.add_streamer ("file://data.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"))
```

(continues on next page)

(continued from previous page)

```
                {
                    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]);
                }
                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 examples
{
    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)));
        }
    }
}
```

### 5.3.4 C# Downsample Data

```
using System;

using brainflow;
using brainflow.math;


namespace examples
{
    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_board_data ();
            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 examples
{
    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]), (int)WaveletTypes.DB4, 1,␣
↪(int)WaveletExtensionTypes.SYMMETRIC);
                // 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, (int)WaveletTypes.DB4,
↪ 1, (int)WaveletExtensionTypes.SYMMETRIC);
```

(continues on next page)

(continued from previous page)

```
                Console.WriteLine ("Restored wavelet data:");
                Console.WriteLine ("[{0}]", string.Join (", ", restored_data));

                // demo for fft
                // 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)WindowOperations.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 examples
{
    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 (int i = 0; i < eeg_channels.Length; i++)
            {
                DataFilter.detrend (unprocessed_data, eeg_channels[i],␣
↪(int)DetrendOperations.CONSTANT);
                DataFilter.perform_bandstop (unprocessed_data, eeg_channels[i],␣
↪BoardShim.get_sampling_rate (board_id), 48.0, 52.0, 4, (int)FilterTypes.BUTTERWORTH, 0.
↪0);
```

(continues on next page)

(continued from previous page)

```
                DataFilter.perform_bandpass (unprocessed_data, eeg_channels[i],␣
↪BoardShim.get_sampling_rate (board_id), 4.0, 30.0, 4, (int)FilterTypes.BUTTERWORTH, 0.
↪0);
            }
        }
    }
}
```

### 5.3.7 C# Denoising

```
using System;

using brainflow;
using brainflow.math;


namespace examples
{
    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);
            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]);
```

(continues on next page)

(continued from previous page)

```
                        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]), (int)WaveletTypes.BIOR3_9, 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 examples
{
    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);
```

(continues on next page)

(continued from previous page)

```
            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)WindowOperations.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 examples
{
    class EEGMetrics
    {
        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[] 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);
```

(continues on next page)

(continued from previous page)

```
            double[] feature_vector = bands.Item1;
            BrainFlowModelParams model_params = new BrainFlowModelParams␣
↪((int)BrainFlowMetrics.MINDFULNESS, (int)BrainFlowClassifiers.DEFAULT_CLASSIFIER);
            MLModel model = new MLModel (model_params);
            model.prepare ();
            Console.WriteLine ("Score: " + model.predict (feature_vector)[0]);
            model.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];
                }
                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];
```

(continues on next page)

(continued from previous page)

```
                }
                if (args[i].Equals ("--file"))
                {
                    input_params.file = args[i + 1];
                }
            }
            return board_id;
        }
    }
}
```

### 5.3.10 C# ICA

```
using System;
using System.Runtime.Serialization;

using brainflow;
using brainflow.math;


namespace examples
{
    class ICA
    {
        static void Main (string[] args)
        {
            BoardShim.enable_dev_board_logger ();

            int board_id = (int)BoardIds.SYNTHETIC_BOARD;
            BoardDescr board_descr = BoardShim.get_board_descr<BoardDescr> (board_id);
            int[] eeg_channels = board_descr.eeg_channels;
            int channel = eeg_channels[1];

            BrainFlowInputParams input_params = new BrainFlowInputParams ();
            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 (500);
            board_shim.release_session ();

            double[,] ica_data = data.GetRow (channel).Reshape(5, 100);
            Tuple<double[,], double[,], double[,], double[,]> ica = DataFilter.perform_
↪ica (ica_data, 2);
        }
    }
}
```

## 5.4 C++

To compile examples below for Linux or MacOS run:

```
cd cpp_package/examples/get_data
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 ..
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}}")
```

(continues on next page)

(continued from previous page)

```
                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
)

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 (
```

(continues on next page)

(continued from previous page)

```
    markers PUBLIC
    # for some systems(ubuntu for example) order matters
    ${BrainflowPath}
    ${MLModulePath}
    ${DataHandlerPath}
    ${BoardControllerPath}
)

add_executable (
    get_data_twice
    src/get_data_twice.cpp
)

target_include_directories (
    get_data_twice PUBLIC
    ${brainflow_INCLUDE_DIRS}
)

target_link_libraries (
    get_data_twice PUBLIC
    # for some systems(ubuntu for example) order matters
    ${BrainflowPath}
    ${MLModulePath}
    ${DataHandlerPath}
    ${BoardControllerPath}
)

add_executable (
    get_data_muse
    src/get_data_muse.cpp
)

target_include_directories (
    get_data_muse PUBLIC
    ${brainflow_INCLUDE_DIRS}
)

target_link_libraries (
    get_data_muse PUBLIC
    # for some systems(ubuntu for example) order matters
    ${BrainflowPath}
    ${MLModulePath}
    ${DataHandlerPath}
    ${BoardControllerPath}
)


add_executable (
    multiple_streamers
    src/multiple_streamers.cpp
)
```

(continues on next page)

(continued from previous page)

```
target_include_directories (
    multiple_streamers PUBLIC
    ${brainflow_INCLUDE_DIRS}
)

target_link_libraries (
    multiple_streamers 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);


int main (int argc, char *argv[])
{
    BoardShim::enable_dev_board_logger ();

    BoardShim::get_board_presets (-1);
    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 ();

#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;
                *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"))
        {
```

(continues on next page)

(continued from previous page)

```
            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-address-aux"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_address_aux = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-address-anc"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_address_anc = 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 ("--ip-port-aux"))
        {
```

(continues on next page)

(continued from previous page)

```
            if (i + 1 < argc)
            {
                i++;
                params->ip_port_aux = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--ip-port-anc"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->ip_port_anc = 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"))
        {
```

(continues on next page)

(continued from previous page)

```
            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 (std::string (argv[i]) == std::string ("--file-aux"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->file_aux = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--file-anc"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->file_anc = std::string (argv[i]);
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
        if (std::string (argv[i]) == std::string ("--master-board"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->master_board = std::stoi (std::string (argv[i]));
            }
            else
            {
                std::cerr << "missed argument" << std::endl;
                return false;
            }
        }
    }
    if (!board_id_found)
```

(continues on next page)

(continued from previous page)

```
    {
        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
    {
        board->prepare_session ();
        board->start_stream ();

        for (int i = 1; i < 5; i++)
        {
            board->insert_marker (i);
#ifdef _WIN32
```

(continues on next page)

(continued from previous page)

```
            Sleep (2000);
#else
            sleep (2);
#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)
            {
                i++;
                params->ip_address = 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 ("--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]));
```

(continues on next page)

(continued from previous page)

```
            }
            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"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->file = 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 ("--master-board"))
        {
            if (i + 1 < argc)
            {
                i++;
                params->master_board = std::stoi (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;
}
```

### 5.4.4 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 ();
```

(continues on next page)

(continued from previous page)

```
    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;
    }
    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
```

(continues on next page)

(continued from previous page)

```
#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);
#endif

        board->stop_stream ();
        BrainFlowArray<double, 2> data = board->get_board_data ();
        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:
```

(continues on next page)

(continued from previous page)

```
                    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 ();
        }
    }

    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;
```

(continues on next page)

(continued from previous page)

```
}
```

### 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
        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);
```

(continues on next page)

(continued from previous page)

```
        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, (int)WaveletTypes::DB3,␣
↪3);
            // 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, (int)WaveletTypes::DB3, 3);

            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;

            // demo for fft
            // data count must be power of 2 for fft!
            int fft_len = 0;
            std::complex<double> *fft_data =
                DataFilter::perform_fft (data.get_address (eeg_channels[i]), data_count,
                    (int)WindowOperations::NO_WINDOW, &fft_len);
            std::cout << "FFT coeffs:" << std::endl;
            for (int i = 0; i < fft_len; i++)
```

(continues on next page)

(continued from previous page)

```
            {
                std::cout << fft_data[i] << " ";
            }
            std::cout << std::endl;
            int restored_len = 0;
            double *restored_from_fft_data =
                DataFilter::perform_ifft (fft_data, fft_len, &restored_len);
            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>
#endif
```

(continues on next page)

(continued from previous page)

```
#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), 50.0,
↪ 3,
                        (int)FilterTypes::BUTTERWORTH_ZERO_PHASE, 0);
                    break;
                case 1:
                    DataFilter::perform_highpass (data.get_address (eeg_channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id), 3.0,␣
↪5,
```

(continues on next page)

(continued from previous page)

```
                        (int)FilterTypes::CHEBYSHEV_TYPE_1_ZERO_PHASE, 1);
                    break;
                case 2:
                    DataFilter::perform_bandpass (data.get_address (eeg_channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id), 3.0,␣
↪45.0, 3,
                        (int)FilterTypes::BESSEL_ZERO_PHASE, 0);
                    break;
                case 3:
                    DataFilter::perform_bandstop (data.get_address (eeg_channels[i]),
                        data.get_size (1), BoardShim::get_sampling_rate (board_id), 48.0,
↪ 62.0, 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
```

(continues on next page)

(continued from previous page)

```
#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
        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;
                default:
                    // if moving average and moving median dont work well for your␣
↪signal you can
```

(continues on next page)

(continued from previous page)

```
                    // try wavelet based denoising, feel free to try different wavelet␣
↪functions and
                    // decomposition levels
                    DataFilter::perform_wavelet_denoising (data.get_address (eeg_
↪channels[i]),
                        data.get_size (1), (int)WaveletTypes::BIOR3_9, 3,
                        (int)WaveletDenoisingTypes::SURESHRINK, (int)ThresholdTypes::
↪HARD,
                        (int)WaveletExtensionTypes::SYMMETRIC,
                        (int)NoiseEstimationLevelTypes::FIRST_LEVEL);
                    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[])
{
```

(continues on next page)

(continued from previous page)

```
    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 (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;
        int psd_len = 0;
        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)WindowOperations::HANNING, &psd_len);
        // calc band power
        double band_power_alpha = DataFilter::get_band_power (psd, psd_len, 7.0, 13.0);
        double band_power_beta = DataFilter::get_band_power (psd, psd_len, 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 ());
```

(continues on next page)

(continued from previous page)

```
        res = err.exit_code;
        if (board->is_prepared ())
        {
            board->release_session ();
        }
    }

    delete board;

    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
    {
```

(continues on next page)

(continued from previous page)

```
        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
        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);

        struct BrainFlowModelParams mindfulness_params (
            (int)BrainFlowMetrics::MINDFULNESS, (int)BrainFlowClassifiers::DEFAULT_
↪CLASSIFIER);
        MLModel mindfulness_model (mindfulness_params);
        mindfulness_model.prepare ();
        std::cout << "Mindfulness :" << mindfulness_model.predict (bands.first, 5)[0] <<␣
↪std::endl;
        mindfulness_model.release ();

        struct BrainFlowModelParams restfulness_params (
            (int)BrainFlowMetrics::RESTFULNESS, (int)BrainFlowClassifiers::DEFAULT_
↪CLASSIFIER);
        MLModel restfulness_model (restfulness_params);
        restfulness_model.prepare ();
        std::cout << "Restfulness :" << restfulness_model.predict (bands.first, 5)[0] <<␣
↪std::endl;
        restfulness_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;
```

(continues on next page)

(continued from previous page)

```
    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)
            {
                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"))
        {
```

(continues on next page)

(continued from previous page)

```
            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
            {
                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"))
        {
```

(continues on next page)

(continued from previous page)

```
            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;
            }
        }
    }
    if (!board_id_found)
    {
        std::cerr << "board id is not provided" << std::endl;
        return false;
    }
    return true;
}
```

### 5.4.11 C++ ICA

```
#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;
    std::vector<int> eeg_channels = BoardShim::get_eeg_channels (board_id);
    int channel_to_use = eeg_channels[4];
    // 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 (500, (int)BrainFlowPresets::DEFAULT_PRESET);
        board->release_session ();

        BrainFlowArray<double, 2> data_reshaped (data.get_address (channel_to_use), 5,␣
↪100);
        std::tuple<BrainFlowArray<double, 2>, BrainFlowArray<double, 2>, BrainFlowArray
↪<double, 2>,
            BrainFlowArray<double, 2>>
            returned_matrixes = DataFilter::perform_ica (data_reshaped, 2);
        std::cout << std::get<3> (returned_matrixes) << 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;
}
```

## 5.5 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$get_current_board_data(as.integer(250))
data <- np$ascontiguousarray(data)
board_shim$release_session()

brainflow_python$DataFilter$write_file(data, "test.csv", "w")
data_restored <- brainflow_python$DataFilter$read_file("test.csv")
print(data_restored)
```

### 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))
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$get_psd_welch(numpy_data, as.integer(nfft), as.
↪integer(nfft / 2),
    sampling_rate, brainflow_python$WindowOperations$BLACKMAN_HARRIS$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)
data <- np$ascontiguousarray(data)
eeg_channels <- np$ascontiguousarray(c(eeg_channels))
bands <- brainflow_python$DataFilter$get_avg_band_powers(data, eeg_channels, sampling_
↪rate, TRUE)
feature_vector <- np$array(bands[[1]])

model_params <- brainflow_python$BrainFlowModelParams(brainflow_python$BrainFlowMetrics
↪$MINDFULNESS$value, brainflow_python$BrainFlowClassifiers$DEFAULT_CLASSIFIER$value)
model <- brainflow_python$MLModel(model_params)
```

(continues on next page)

(continued from previous page)

```
model$prepare()
score <- model$predict(feature_vector)
model$release()
```

### 5.5.9 R ICA

```
library(brainflow)

params <- brainflow_python$BrainFlowInputParams()
board_shim <- brainflow_python$BoardShim(brainflow_python$BoardIds$SYNTHETIC_BOARD$value,
↪ params)
board_shim$prepare_session()
board_shim$start_stream()
Sys.sleep(time = 10)
board_shim$stop_stream()
data <- board_shim$get_board_data(as.integer(500))
board_shim$release_session()
eeg_channels <- brainflow_python$BoardShim$get_eeg_channels(as.integer(-1))
my_channel = eeg_channels[2]
numpy_data <- np$array(data[my_channel,])
numpy_data <- numpy_data$reshape(as.integer(5),as.integer(100))
ica <- brainflow_python$DataFilter$perform_ica(numpy_data, as.integer(2))
```

## 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);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
board_shim.prepare_session();
board_shim.add_streamer('file://data_default.csv:w', preset);
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
data = board_shim.get_current_board_data(10, preset);
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);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(2);
board_shim.insert_marker(1, preset);
pause(2);
board_shim.stop_stream();
data = board_shim.get_board_data(board_shim.get_board_data_count(preset), preset);
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);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(2)
board_shim.stop_stream()
data = board_shim.get_current_board_data(20, preset);
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);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
sampling_rate = BoardShim.get_sampling_rate(int32(BoardIds.SYNTHETIC_BOARD), preset);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
data = board_shim.get_current_board_data(256, preset);
```

(continues on next page)

(continued from previous page)

```
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIds.SYNTHETIC_BOARD), preset);
% 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,␣
↪int32(WaveletTypes.DB3), 3, int32(WaveletExtensionTypes.SYMMETRIC));
restored_data = DataFilter.perform_inverse_wavelet_transform(wavelet_data, wavelet_
↪lenghts, size(original_data, 2), int32(WaveletTypes.DB3), 3,␣
↪int32(WaveletExtensionTypes.SYMMETRIC));
% fft for first eeg channel %
fft_data = DataFilter.perform_fft(original_data, int32(WindowOperations.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);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(5);
board_shim.stop_stream();
data = board_shim.get_current_board_data(64, preset);
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIds.SYNTHETIC_BOARD), preset);
% 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), preset);
filtered_data = DataFilter.perform_lowpass(original_data, sampling_rate, 50.0, 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);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(5);
```

(continues on next page)

(continued from previous page)

```
board_shim.stop_stream();
data = board_shim.get_current_board_data(64, preset);
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIds.SYNTHETIC_BOARD), preset);
% 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, int32(WaveletTypes.DB3),
↪ 3, int32(WaveletDenoisingTypes.SURESHRINK), int32(ThresholdTypes.HARD),␣
↪int32(WaveletExtensionTypes.SYMMETRIC), int32(NoiseEstimationLevelTypes.FIRST_LEVEL));
```

### 5.6.7 Matlab Band Power

```
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIds.SYNTHETIC_BOARD), params);
board_id = int32(BoardIds.SYNTHETIC_BOARD);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
board_descr = BoardShim.get_board_descr(board_id, preset);
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.get_board_data_count(preset), preset);
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(WindowOperations.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');
MLModel.set_log_file('brainflow_ml.log');
BoardShim.enable_dev_board_logger();

params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIds.SYNTHETIC_BOARD), params);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
```

(continues on next page)

(continued from previous page)

```
sampling_rate = BoardShim.get_sampling_rate(int32(BoardIds.SYNTHETIC_BOARD), preset);
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.get_board_data_count(preset), preset);
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIds.SYNTHETIC_BOARD), preset);
[avgs, stddevs] = DataFilter.get_avg_band_powers(data, eeg_channels, sampling_rate,␣
↪true);
feature_vector = avgs;

model_params = BrainFlowModelParams(int32(BrainFlowMetrics.RESTFULNESS),␣
↪int32(BrainFlowClassifiers.DEFAULT_CLASSIFIER));
model = MLModel(model_params);
model.prepare();
score = model.predict(feature_vector);
model.release();
```

### 5.6.9 Matlab ICA

```
params = BrainFlowInputParams();
board_shim = BoardShim(int32(BoardIds.SYNTHETIC_BOARD), params);
preset = int32(BrainFlowPresets.DEFAULT_PRESET);
board_shim.prepare_session();
board_shim.start_stream(45000, '');
pause(10);
board_shim.stop_stream();
data = board_shim.get_board_data(500, preset);
board_shim.release_session();

eeg_channels = BoardShim.get_eeg_channels(int32(BoardIds.SYNTHETIC_BOARD), preset);
selected_channel = eeg_channels(4);
original_data = data(selected_channel, :);
original_data = transpose(reshape(original_data, [100, 5]));
[w,k,a,s] = DataFilter.perform_ica(original_data, 2);
```

## 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)
```

(continues on next page)

(continued from previous page)

```
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.add_streamer("file://data_default.csv:w", 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")
```

(continues on next page)

(continued from previous page)

```
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()
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(256, 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, BrainFlow.DB3, 2,␣
↪BrainFlow.SYMMETRIC)
restored_wavelet_data = BrainFlow.perform_inverse_wavelet_transform(wavelet_data,␣
↪length(data_first_channel),

↪DB3, 2, BrainFlow.SYMMETRIC)

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, 50.0, 3, BrainFlow.
↪BUTTERWORTH, 0.0)
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, 3.0, 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, 3.0, 50.0, 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, 48.0, 52.0, 3, BrainFlow.
↪BESSEL, 0.0)
println("After BandStop Filter")
println(data_fourth_channel)
```

(continues on next page)

(continued from previous page)

```
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(256, 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_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, BrainFlow.BIOR3_9, 3, BrainFlow.
↪SURESHRINK,
                                                                    BrainFlow.HARD,␣
↪BrainFlow.SYMMETRIC, BrainFlow.FIRST_LEVEL)
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)
```

(continues on next page)

(continued from previous page)

```
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 = bands[1]
println(feature_vector)

model_params = BrainFlowModelParams(BrainFlow.MINDFULNESS, BrainFlow.DEFAULT_CLASSIFIER)
BrainFlow.prepare(model_params)
print(BrainFlow.predict(feature_vector, model_params))
BrainFlow.release(model_params)
```

### 5.7.9 Julia ICA

```
using BrainFlow

# enable logs
BrainFlow.enable_dev_logger(BrainFlow.BOARD_CONTROLLER)
BrainFlow.enable_dev_logger(BrainFlow.DATA_HANDLER)

params = BrainFlowInputParams()
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_board_data(500, board_shim)
BrainFlow.release_session(board_shim)

eeg_channels = BrainFlow.get_eeg_channels(BrainFlow.SYNTHETIC_BOARD)
data_first_channel = data[eeg_channels[1], :]

my_data = transpose(reshape(data_first_channel, (100, 5)))

ica = BrainFlow.perform_ica(my_data, 2)
```

## 5.8 Rust

### 5.8.1 Rust Get Data from a Board

```
use std::{thread, time::Duration};

use brainflow::{board_shim, brainflow_input_params::BrainFlowInputParamsBuilder,␣
↪BoardIds, BrainFlowPresets,};
```

(continues on next page)

(continued from previous page)

```
fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let params = BrainFlowInputParamsBuilder::default().build();
    let board = board_shim::BoardShim::new(BoardIds::SyntheticBoard, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let data = board.get_board_data(Some(10), BrainFlowPresets::DefaultPreset).unwrap();
    board.release_session().unwrap();

    println!("{}", data.len());
    println!("{:?}", data);
}
```

### 5.8.2 Rust Markers

```
use std::thread;
use std::time::Duration;

use brainflow::board_shim::BoardShim;
use brainflow::brainflow_input_params::BrainFlowInputParamsBuilder;
use brainflow::BoardIds;
use brainflow::BrainFlowPresets;

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let params = BrainFlowInputParamsBuilder::default().build();
    let board = BoardShim::new(BoardIds::SyntheticBoard, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "file://data.csv:w").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.insert_marker(1.0, BrainFlowPresets::DefaultPreset).unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let data = board.get_board_data(Some(10), BrainFlowPresets::DefaultPreset).unwrap();
    board.release_session().unwrap();

    println!("{:?}", data);
}
```

### 5.8.3 Rust Read Write File

```
use std::{env, fs, thread, time::Duration};

use brainflow::{
    board_shim, brainflow_input_params::BrainFlowInputParamsBuilder, data_filter,␣
↪BoardIds, BrainFlowPresets,
};

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let params = BrainFlowInputParamsBuilder::default().build();
    let board = board_shim::BoardShim::new(BoardIds::SyntheticBoard, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let data = board.get_board_data(Some(10), BrainFlowPresets::DefaultPreset).unwrap();
    board.release_session().unwrap();

    let mut tmp_dir = env::temp_dir();
    tmp_dir.push("brainflow_tests");
    tmp_dir.push("rust");
    fs::create_dir_all(&tmp_dir).unwrap();
    tmp_dir.push("read-write_file.csv");

    let filename = tmp_dir.to_str().unwrap();

    dbg!(&data);
    data_filter::write_file(&data, filename, "w").unwrap();
    let read_data = data_filter::read_file(filename).unwrap();
    dbg!(read_data);
}
```

### 5.8.4 Rust Transforms

```
use std::{thread, time::Duration};

use brainflow::{
    board_shim, brainflow_input_params::BrainFlowInputParamsBuilder, data_filter,␣
↪BoardIds,
    WindowOperations, WaveletTypes, WaveletExtensionTypes, BrainFlowPresets,
};
use ndarray::s;

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();

    let board_id = BoardIds::SyntheticBoard;
    let eeg_channels = board_shim::get_eeg_channels(board_id, BrainFlowPresets::
↪DefaultPreset).unwrap();
```

(continues on next page)

(continued from previous page)

```
    println!("{:?}", eeg_channels);

    let params = BrainFlowInputParamsBuilder::default().build();
    let board = board_shim::BoardShim::new(board_id, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let mut data = board.get_board_data(Some(64), BrainFlowPresets::DefaultPreset).
↪unwrap();
    board.release_session().unwrap();

    let data_len = data.slice(s![0, ..]).len();

    let fft_data = data_filter::perform_fft(
        data.slice_mut(s![eeg_channels[0], ..])
            .as_slice_mut()
            .unwrap(),
        WindowOperations::BlackmanHarris,
    )
    .unwrap();
    let restored_fft = data_filter::perform_ifft(&fft_data, data_len).unwrap();
    println!("{:?}", restored_fft);

    println!("{:?}", data.slice(s![1, ..]));
    let wavelet_data = data_filter::perform_wavelet_transform(
        data.slice_mut(s![eeg_channels[1], ..])
            .as_slice_mut()
            .unwrap(),
        WaveletTypes::Db3,
        3,
        WaveletExtensionTypes::Symmetric,
    )
    .unwrap();
    let restored_wavelet = data_filter::perform_inverse_wavelet_transform(wavelet_data).
↪unwrap();
    println!("{:?}", restored_wavelet);
}
```

### 5.8.5 Rust Signal Filtering

```
use std::{thread, time::Duration};

use brainflow::{board_shim, brainflow_input_params::BrainFlowInputParamsBuilder,␣
↪BoardIds, BrainFlowPresets,};

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let params = BrainFlowInputParamsBuilder::default().build();
```

(continues on next page)

(continued from previous page)

```
    let board = board_shim::BoardShim::new(BoardIds::SyntheticBoard, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let data = board.get_board_data(Some(10), BrainFlowPresets::DefaultPreset).unwrap();
    board.release_session().unwrap();

    println!("{}", data.len());
    println!("{:?}", data);
}
```

### 5.8.6 Rust Denoising

```
use std::{thread, time::Duration};

use brainflow::{
    board_shim, brainflow_input_params::BrainFlowInputParamsBuilder, data_filter,␣
↪AggOperations,
    BoardIds, WaveletTypes, WaveletExtensionTypes, WaveletDenoisingTypes, ThresholdTypes,
↪ NoiseEstimationLevelTypes,
    BrainFlowPresets,
};
use ndarray::s;

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let board_id = BoardIds::SyntheticBoard;
    let eeg_channels = board_shim::get_eeg_channels(board_id, BrainFlowPresets::
↪DefaultPreset).unwrap();

    let params = BrainFlowInputParamsBuilder::default().build();
    let board = board_shim::BoardShim::new(board_id, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    board.add_streamer("file://data.csv:w", BrainFlowPresets::DefaultPreset).unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let mut data = board.get_board_data(Some(128), BrainFlowPresets::DefaultPreset).
↪unwrap();
    board.release_session().unwrap();

    println!("{:?}", data.slice(s![eeg_channels[0], ..]));
    data_filter::perform_rolling_filter(
        data.slice_mut(s![eeg_channels[0], ..])
            .as_slice_mut()
            .unwrap(),
        3,
```

(continues on next page)

(continued from previous page)

```
        AggOperations::Mean,
    )
    .unwrap();
    println!("{:?}", data.slice(s![eeg_channels[0], ..]));

    println!("{:?}", data.slice(s![eeg_channels[1], ..]));
    data_filter::perform_wavelet_denoising(
        data.slice_mut(s![eeg_channels[1], ..])
            .as_slice_mut()
            .unwrap(),
        WaveletTypes::Db3,
        3,
        WaveletDenoisingTypes::Sureshrink,
        ThresholdTypes::Hard,
        WaveletExtensionTypes::Symmetric,
        NoiseEstimationLevelTypes::FirstLevel,
    )
    .unwrap();
    println!("{:?}", data.slice(s![eeg_channels[1], ..]));
}
```

### 5.8.7 Rust Band Power

```
use brainflow::data_filter::Band;
use std::{thread, time::Duration};

use brainflow::{
    board_shim, brainflow_input_params::BrainFlowInputParamsBuilder, data_filter,␣
↪BoardIds,
    DetrendOperations, WindowOperations, BrainFlowPresets,
};
use ndarray::s;

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let board_id = BoardIds::SyntheticBoard;
    let eeg_channels = board_shim::get_eeg_channels(board_id, BrainFlowPresets::
↪DefaultPreset).unwrap();
    let sampling_rate = board_shim::get_sampling_rate(board_id, BrainFlowPresets::
↪DefaultPreset).unwrap();
    let nfft = data_filter::get_nearest_power_of_two(sampling_rate).unwrap();

    let params = BrainFlowInputParamsBuilder::default().build();
    let board = board_shim::BoardShim::new(board_id, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let mut data = board.get_board_data(None, BrainFlowPresets::DefaultPreset).unwrap();
```

(continues on next page)

(continued from previous page)

```
    board.release_session().unwrap();

    data_filter::detrend(
        data.slice_mut(s![eeg_channels[1], ..])
            .as_slice_mut()
            .unwrap(),
        DetrendOperations::Linear,
    )
    .unwrap();
    let mut psd = data_filter::get_psd_welch(
        data.slice_mut(s![eeg_channels[1], ..])
            .as_slice_mut()
            .unwrap(),
        nfft,
        nfft / 2,
        sampling_rate,
        WindowOperations::BlackmanHarris,
    )
    .unwrap();
    let band_power_alpha = data_filter::get_band_power(&mut psd, Band { freq_start: 8.0,␣
↪freq_stop: 13.0 }).unwrap();
    let band_power_beta = data_filter::get_band_power(&mut psd, Band { freq_start: 14.0,␣
↪freq_stop: 30.0 }).unwrap();
    println!(
        "band_power_alpha / band_power_beta = {}",
        band_power_alpha / band_power_beta
    );
}
```

### 5.8.8 Rust EEG Metrics

```
use std::{thread, time::Duration};

use brainflow::{
    board_shim, brainflow_input_params::BrainFlowInputParamsBuilder,
    brainflow_model_params::BrainFlowModelParamsBuilder, data_filter, ml_model, BoardIds,
    BrainFlowClassifiers, BrainFlowMetrics, BrainFlowPresets,
};

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();
    let params = BrainFlowInputParamsBuilder::default().build();
    let board_id = BoardIds::SyntheticBoard;
    let board = board_shim::BoardShim::new(board_id, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(5));
    board.stop_stream().unwrap();
    let data = board.get_board_data(None, BrainFlowPresets::DefaultPreset).unwrap();
```

(continues on next page)

(continued from previous page)

```
    board.release_session().unwrap();

    let eeg_channels = board_shim::get_eeg_channels(board_id, BrainFlowPresets::
↪DefaultPreset).unwrap();
    let sampling_rate = board_shim::get_sampling_rate(board_id, BrainFlowPresets::
↪DefaultPreset).unwrap();
    let bands =
        data_filter::get_avg_band_powers(data, eeg_channels, sampling_rate, true).
↪unwrap();
    let mut feature_vector = bands.0;
    println!("feature_vector: {:?}", feature_vector);

    let model_params = BrainFlowModelParamsBuilder::new()
        .metric(BrainFlowMetrics::Mindfulness)
        .classifier(BrainFlowClassifiers::DefaultClassifier)
        .build();
    let mindfulness = ml_model::MlModel::new(model_params).unwrap();
    mindfulness.prepare().unwrap();
    println!(
        "Mindfulness: {:?}",
        mindfulness.predict(&mut feature_vector)
    );
    mindfulness.release().unwrap();
}
```

### 5.8.9 Rust ICA

```
use ndarray::Array2;
use std::{thread, time::Duration};

use brainflow::{
    board_shim, brainflow_input_params::BrainFlowInputParamsBuilder, data_filter,␣
↪BoardIds, BrainFlowPresets,
};
use ndarray::s;

fn main() {
    brainflow::board_shim::enable_dev_board_logger().unwrap();

    let board_id = BoardIds::SyntheticBoard;
    let eeg_channels = board_shim::get_eeg_channels(board_id, BrainFlowPresets::
↪DefaultPreset).unwrap();
    println!("{:?}", eeg_channels);

    let params = BrainFlowInputParamsBuilder::default().build();
    let board = board_shim::BoardShim::new(board_id, params).unwrap();

    board.prepare_session().unwrap();
    board.start_stream(45000, "").unwrap();
    thread::sleep(Duration::from_secs(10));
```

(continues on next page)

(continued from previous page)

```
    board.stop_stream().unwrap();
    let mut data = board.get_board_data(Some(500), BrainFlowPresets::DefaultPreset).
↪unwrap();
    board.release_session().unwrap();

    let my_channel = eeg_channels[4];
    let mut my_data = data.slice_mut(s![my_channel, ..]);
    let data_my_channel = my_data.as_slice_mut().unwrap();
    let ica_data = Array2::from_shape_vec((5, 100), data_my_channel.to_vec()).unwrap();
    let _ica = data_filter::perform_ica(ica_data, 2).unwrap();
}
```

## 5.9 Typescript

### 5.9.1 Typescript Get Data from a Board

```
import {BoardIds, BoardShim} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const board = new BoardShim (BoardIds.SYNTHETIC_BOARD, {});
    board.prepareSession();
    board.startStream();
    await sleep (3000);
    board.stopStream();
    const data = board.getBoardData();
    board.releaseSession();
    console.info('Data');
    console.info(data);
}

runExample ();
```

### 5.9.2 Typescript Markers

```
import {BoardIds, BoardShim} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}
```

(continues on next page)

(continued from previous page)

```
async function runExample (): Promise<void>
{
    const board = new BoardShim (BoardIds.SYNTHETIC_BOARD, {});
    board.prepareSession();
    board.startStream();
    await sleep (1000);
    board.insertMarker(1);
    await sleep (1000);
    board.stopStream();
    const data = board.getBoardData();
    board.releaseSession();
    console.info('Data');
    console.info(data);
}

runExample ();
```

### 5.9.3 Typescript Read Write File

```
import {BoardIds, BoardShim, DataFilter} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const board = new BoardShim (BoardIds.SYNTHETIC_BOARD, {});
    board.prepareSession();
    board.startStream();
    await sleep (3000);
    board.stopStream();
    const data = board.getCurrentBoardData(10);
    board.releaseSession()
    console.info('Data');
    console.info(data);
    DataFilter.writeFile(data, 'test.csv', 'w');
    const dataRestored = DataFilter.readFile('test.csv');
    console.info('Data restored');
    console.info(dataRestored);
    DataFilter.writeFile(dataRestored, 'test2.csv', 'w');
}

runExample ();
```

### 5.9.4 Typescript Downsample Data

```
import {AggOperations, BoardIds, BoardShim, DataFilter} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const boardId = BoardIds.SYNTHETIC_BOARD;
    const board = new BoardShim (boardId, {});
    board.prepareSession();
    board.startStream();
    await sleep (3000);
    board.stopStream();
    const data = board.getCurrentBoardData(10);
    board.releaseSession()
    const eegChannels = BoardShim.getEegChannels(boardId);
    const oldData = data[eegChannels[0]];
    console.info(oldData);
    const newData = DataFilter.performDownsampling(oldData, 3, AggOperations.MEAN);
    console.info(newData);
}

runExample ();
```

### 5.9.5 Typescript Transforms

```
import {
    BoardIds,
    BoardShim,
    DataFilter,
    WaveletExtensionTypes,
    WaveletTypes,
    WindowOperations
} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const boardId = BoardIds.SYNTHETIC_BOARD;
    const board = new BoardShim (boardId, {});
    board.prepareSession();
    board.startStream();
    await sleep (3000);
```

(continues on next page)

(continued from previous page)

```
    board.stopStream();
    const data = board.getCurrentBoardData(64);
    board.releaseSession()
    const eegChannels = BoardShim.getEegChannels(boardId);
    const oldData = data[eegChannels[0]];
    console.info(oldData);
    const fftData = DataFilter.performFft(oldData, WindowOperations.NO_WINDOW);
    console.info(fftData);
    const newData = DataFilter.performIfft(fftData);
    console.info(newData);
    const waveletData = DataFilter.performWaveletTransform(
        newData, WaveletTypes.DB2, 2, WaveletExtensionTypes.SYMMETRIC);
    const restoredData = DataFilter.performInverseWaveletTransform(
        waveletData, newData.length, WaveletTypes.DB2, 2, WaveletExtensionTypes.
↪SYMMETRIC);
    console.info(restoredData);
}

runExample ();
```

### 5.9.6 Typescript Signal Filtering

```
import {BoardIds, BoardShim, DataFilter, FilterTypes} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const boardId = BoardIds.SYNTHETIC_BOARD;
    const board = new BoardShim (boardId, {});
    board.prepareSession();
    board.startStream();
    await sleep (3000);
    board.stopStream();
    const data = board.getBoardData();
    board.releaseSession()
    console.info('Data');
    console.info(data);
    const eegChannels = BoardShim.getEegChannels(boardId);
    const samplingRate = BoardShim.getSamplingRate(boardId);
    DataFilter.performBandPass(
        data[eegChannels[0]], samplingRate, 3.0, 20.0, 2, FilterTypes.BUTTERWORTH_ZERO_
↪PHASE, 0.0);
    DataFilter.performBandStop(
        data[eegChannels[0]], samplingRate, 48.0, 52.0, 2, FilterTypes.BUTTERWORTH_ZERO_
↪PHASE, 0.0);
    DataFilter.performLowPass(
```

(continues on next page)

(continued from previous page)

```
        data[eegChannels[0]], samplingRate, 48.0, 2, FilterTypes.BUTTERWORTH_ZERO_PHASE,␣
↪0.0);
    DataFilter.performHighPass(
        data[eegChannels[0]], samplingRate, 3.0, 2, FilterTypes.BUTTERWORTH_ZERO_PHASE,␣
↪0.0);
}

runExample ();
```

### 5.9.7 Typescript Denoising

```
import {BoardIds, BoardShim, DataFilter, WaveletTypes} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const boardId = BoardIds.SYNTHETIC_BOARD;
    const board = new BoardShim (boardId, {});
    board.prepareSession();
    board.startStream();
    await sleep (5000);
    board.stopStream();
    const data = board.getBoardData();
    board.releaseSession()
    const eegChannels = BoardShim.getEegChannels(boardId);
    const oldData = data[eegChannels[0]];
    console.info(oldData);
    DataFilter.performWaveletDenoising(oldData, WaveletTypes.DB2, 2);
    console.info(oldData);
}

runExample ();
```

### 5.9.8 Typescript Band Power

```
import {BoardIds, BoardShim, DataFilter, WindowOperations} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const boardId = BoardIds.SYNTHETIC_BOARD;
```

(continues on next page)

(continued from previous page)

```
    const board = new BoardShim (boardId, {});
    board.prepareSession();
    board.startStream();
    await sleep (5000);
    board.stopStream();
    const data = board.getCurrentBoardData(128);
    board.releaseSession()
    const eegChannels = BoardShim.getEegChannels(boardId);
    const samplingRate = BoardShim.getSamplingRate(boardId);
    const oldData = data[eegChannels[1]];
    const psd = DataFilter.getPsdWelch(oldData, 64, 0.5, samplingRate, WindowOperations.
↪HAMMING);
    const alpha = DataFilter.getBandPower(psd, 7.0, 13.0);
    const beta = DataFilter.getBandPower(psd, 14.0, 30.0);
    console.info(alpha / beta);
}

runExample ();
```

### 5.9.9 Typescript EEG Metrics

```
import {
    BoardIds,
    BoardShim,
    BrainFlowClassifiers,
    BrainFlowMetrics,
    DataFilter,
    MLModel
} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const boardId = BoardIds.SYNTHETIC_BOARD;
    const board = new BoardShim (boardId, {});
    board.prepareSession();
    board.startStream();
    await sleep (3000);
    board.stopStream();
    const data = board.getBoardData();
    board.releaseSession()
    console.info(data);
    const eegChannels = BoardShim.getEegChannels(boardId);
    const samplingRate = BoardShim.getSamplingRate(boardId);
    const bands = DataFilter.getAvgBandPowers(data, eegChannels, samplingRate, true);
    const model =
```

(continues on next page)

(continued from previous page)

```
        new MLModel (BrainFlowMetrics.RESTFULNESS, BrainFlowClassifiers.DEFAULT_
↪CLASSIFIER, {});
    model.prepare();
    console.info(model.predict(bands[0]));
    model.release();
}

runExample ();
```

### 5.9.10 Typescript ICA

```
import {BoardIds, BoardShim, DataFilter} from 'brainflow';

function sleep (ms: number)
{
    return new Promise ((resolve) => { setTimeout (resolve, ms); });
}

async function runExample (): Promise<void>
{
    const boardId = BoardIds.SYNTHETIC_BOARD;
    const board = new BoardShim (boardId, {});
    board.prepareSession();
    board.startStream();
    await sleep (10000);
    board.stopStream();
    const data = board.getCurrentBoardData(500);
    board.releaseSession()
    const eegChannels = BoardShim.getEegChannels(boardId);
    const eegData = data[eegChannels[0]];
    const eeg2D: number[][] = [];
    while (eegData.length)
        eeg2D.push(eegData.splice(0, 100));
    const icaData = DataFilter.performIca(eeg2D, 2, [0, 1, 2, 3, 4]);
    console.info(icaData[3]);
}

runExample ();
```

## 5.10 Notebooks

### 5.10.1 BrainFlow to MNE Python Notebook

In [1]:
```
import time
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
```

(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
```

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()
```

```
[2024-04-23 08:26:09.796] [board_logger] [info] incoming json: {
    "file": "",
    "file_anc": "",
    "file_aux": "",
    "ip_address": "",
    "ip_address_anc": "",
    "ip_address_aux": "",
    "ip_port": 0,
    "ip_port_anc": 0,
    "ip_port_aux": 0,
    "ip_protocol": 0,
    "mac_address": "",
    "master_board": -100,
    "other_info": "",
    "serial_number": "",
    "serial_port": "",
    "timeout": 0
}
```

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=2502
```

```
    Range : 0 ... 2501 =      0.000 ...    10.004 secs
```

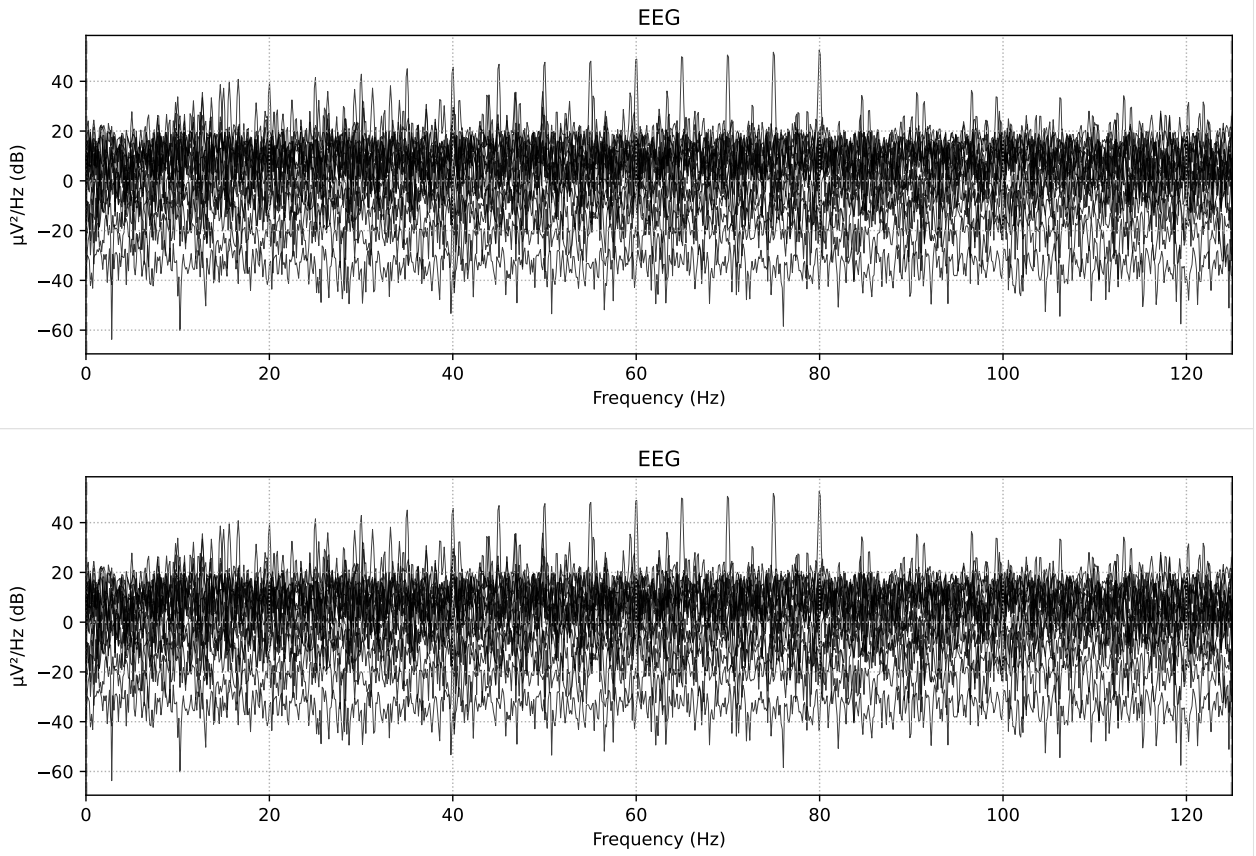
```
Ready.
```

```
NOTE: plot_psd() is a legacy function. New code should use .compute_psd().plot().
```

```
Effective window size : 8.192 (s)
```

```
/tmp/ipykernel_922/1429959458.py:8: RuntimeWarning: Channel locations not available.␣
↪Disabling spatial colors.
  raw.plot_psd(average=False)
```

Out[4]:




### 5.10.2 Denoising Notebook

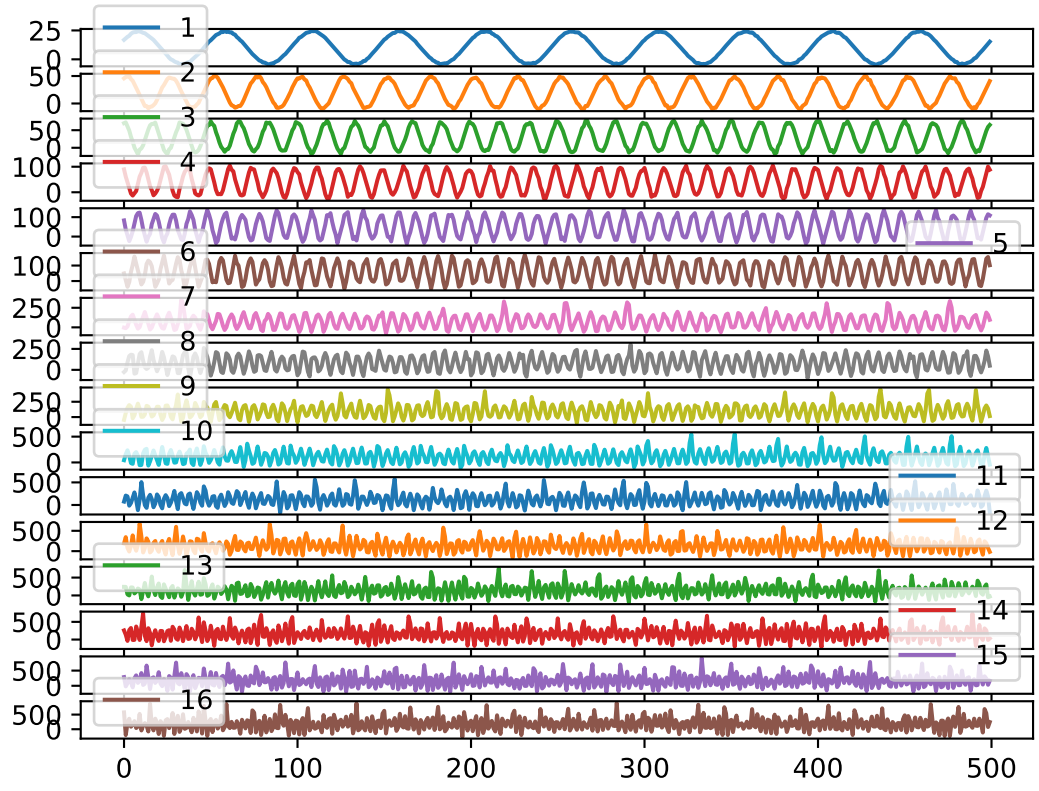
In [1]:

```
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, AggOperations, WaveletTypes,␣
↪NoiseEstimationLevelTypes, WaveletExtensionTypes, ThresholdTypes, WaveletDenoisingTypes
```

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(10)
data = board.get_current_board_data(500)
board.stop_stream()
board.release_session()
```

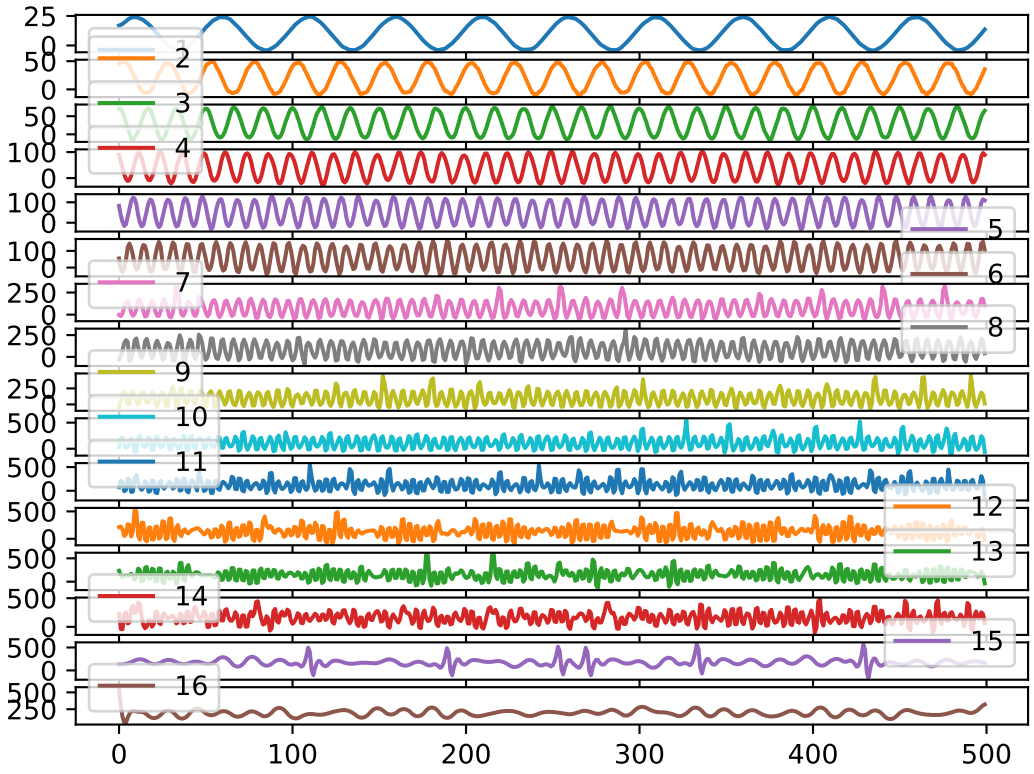
```
[2024-04-23 08:26:24.523] [board_logger] [info] incoming json: {
    "file": "",
    "file_anc": "",
    "file_aux": "",
    "ip_address": "",
    "ip_address_anc": "",
    "ip_address_aux": "",
    "ip_port": 0,
    "ip_port_anc": 0,
    "ip_port_aux": 0,
    "ip_protocol": 0,
    "mac_address": "",
    "master_board": -100,
    "other_info": "",
    "serial_number": "",
    "serial_port": "",
    "timeout": 0
}
```

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 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 parameters
    else:
        DataFilter.perform_wavelet_denoising(data[channel], WaveletTypes.BIOR3_9, 3,␣
↪WaveletDenoisingTypes.SURESHRINK, ThresholdTypes.HARD,
                                             WaveletExtensionTypes.SYMMETRIC,␣
↪NoiseEstimationLevelTypes.FIRST_LEVEL)
```

In [5]:
```
# plot denoised data
df = pd.DataFrame(np.transpose(data))
df[eeg_channels].plot(subplots=True)
plt.show()
```




### 5.10.3 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,␣
↪WindowOperations, DetrendOperations
```

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()
```

(continues on next page)

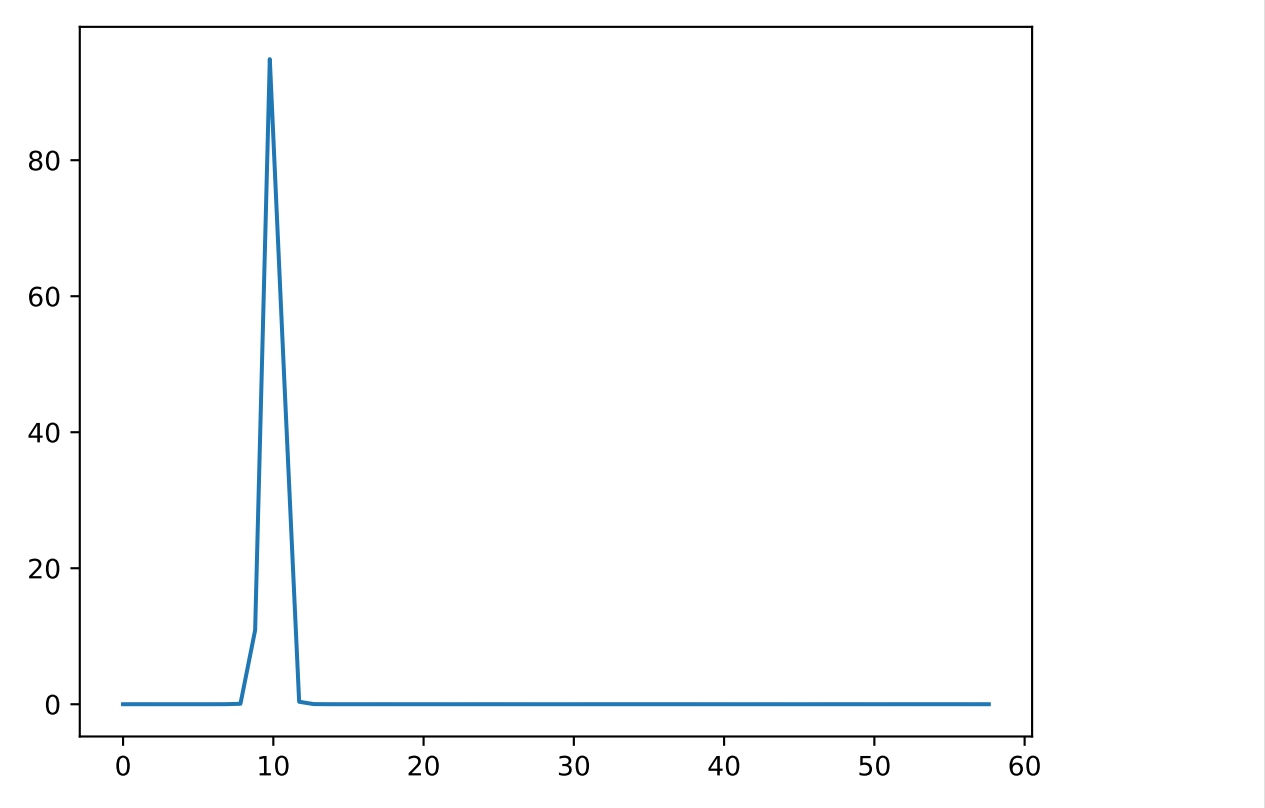
(continued from previous page)

```
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]
```

```
[2024-04-23 08:25:56.268] [board_logger] [info] incoming json: {
    "file": "",
    "file_anc": "",
    "file_aux": "",
    "ip_address": "",
    "ip_address_anc": "",
    "ip_address_aux": "",
    "ip_port": 0,
    "ip_port_anc": 0,
    "ip_port_aux": 0,
    "ip_protocol": 0,
    "mac_address": "",
    "master_board": -100,
    "other_info": "",
    "serial_number": "",
    "serial_port": "",
    "timeout": 0
}
```

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,␣
↪WindowOperations.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: 2479.965613
```

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.

### 6.1.1 Setup

You can build C# binding from source or download precompiled package directly from Nuget.

Here we will provide steps to configure it using precompiled package from Nuget.

- Download the latest NuGet Unity package from NuGetForUnity
- Open your Unity project
- Click on NuGetForUnity.x.x.x.unitypackage and add it into your project
- In the Unity editor, go to NuGet/Manage NuGet Packages
- Search for brainflow and click Install

Now, you are able to use BrainFlow API in your Unity project.

### 6.1.2 Examples

For this demo we will create a simple script to read data from the board.

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;
```

(continues on next page)

(continued from previous page)

```
    // 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 (BrainFlowError 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
            {
                board_shim.release_session();
            }
            catch (BrainFlowError e)
            {
                Debug.Log(e);
            }
            Debug.Log("Brainflow streaming was stopped");
        }
    }
```

(continues on next page)

(continued from previous page)

```
}
```

After building your game for production don't forget to copy *Unmanaged(C++)* libraries to a folder where executable is located.

### 6.1.3 Fixing errors

If you get the error: "Failed to load 'Assets/Packages/brainflow.x.x.x/lib/BoardController32.dll', expected x64 architecture, but was x86 architecture." Or similar error for other native libraries from BrainFlow you should open this library in Unity editor and fix checkboxes for CPU.

- Find Assets/Packages/brainflow.x.x.x/lib/BoardController32.dll inside Unity Editor and click on properties in context menu
- In the properties window, on the left tab, under CPU, select "x86"
- In the properties window, on the right tab, select ONLY the "x86" checkbox
- Restart Unity Editor

## 6.2 Unreal Engine

We provide Unreal Engine Plugin with instructions how to compile and use it. 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

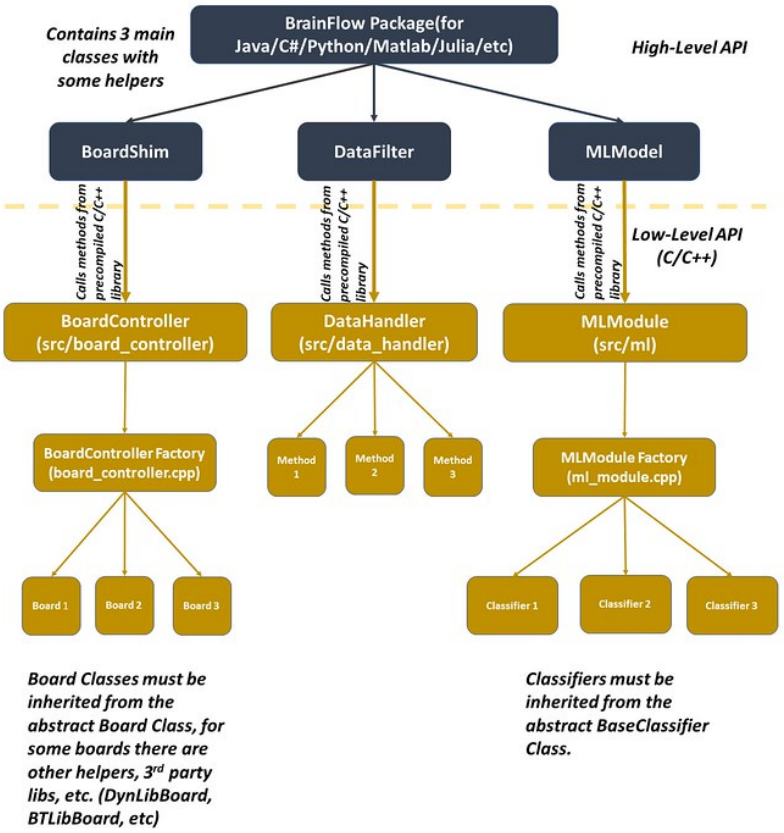
SEVEN

# BRAINFLOW DEV

We encourage contributions to BrainFlow project from individuals at all levels - students, postdocs, academics, industry coders, etc. Knowledge of the domain of signal processing and/or neurotechx is helpful but not required.

If you want to pick up and try an existing improvement project, you will note that we have indicated levels of difficulty with labels. We recommend you to pick a task marked by good first issue label. Most internal hacks will require knowledge of C/C++. Also, we are looking for Rust developers. Knowledge of other languages is useful for binding development.

## 7.1 Navigation



## 7.2 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.

## 7.3 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.

## 7.4 Pull Requests

Just try to briefly explain a goal of this PR.

## 7.5 Instructions to add new boards to BrainFlow

- fork and clone the repo, create a branch other than master
- 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 and helpers like *DynLibBoard, BLELibBoard, etc*
- add information about your board to brainflow_boards.cpp
- add new files to BOARD_CONTROLLER_SRC variable in build.cmake, you may also need to add new directory to *target_include_directories*.
- create a PR

**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.

## 7.6 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
```

## 7.7 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
- reproduce your issue using C++ binding
- run it with debuger and memory checker

Example for Linux, for other OSes it's similar:

```
# Change build type to Debug
python3 tools/build.py --debug --clear-build-dir --num-jobs 8
# Create a test to reproduce your issue in C++, here we will use get_data_demo
cd cpp_package/examples/get_data
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 -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
```

(continues on next page)

(continued from previous page)

```
# 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
```

## 7.8 BrainFlow Emulator

BrainFlow Emulator allows you to run all integration tests for some of supported boards without real hardware. Our CI uses it for test automation. Also, you can run it on your own PC!

Emulators are intended to test BrainFlow code for particular device. Also, some advanced emulators are capable to test very device specific features. BrainFlow users should use Synthetic board or Playback board for development.

## 7.9 Contributors

Andrey1994 is an author and maintainer of BrainFlow project. Full list of developers who can commit directly to this project and merge Pull Requests can be found here.

- Andrey1994
- daniellasry
- philippitts
- xloem
- matthijscox
- mesca
- retiutut
- trobanga
- shirleyzhang867
- Fan1117
- John42506176Linux
- isaacplotkin
- evaesteban
- purplesyringa
- stellarpower
- Sarthak031
- dvaposto
- myd7349
- alexcastillo
- furmanlukasz

- xgdgsc
- serense
- khoinguyen-hub
- fullflyt
- dependabot[bot]

CHAPTER

EIGHT

# ASK HELP

## 8.1 Contact Info, Feature Requests, Issues

- For collaboration requests use contact@brainflow.org
- Join our slack workspace using self-invite page. Use **#askhelp** channel for your questions.
- To report bugs or request features create an issue in our GitHub Page
- We do not provide support for hardware and firmware, for such questions and issues you should contact device manufacturer.
- We also do not provide tech support for applications built on top of BrainFlow, but you are welcome to discuss them with others in **#brainapps** channel.

## 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

CHAPTER

NINE

# PARTNERS, SPONSORS, AND CONTRIBUTORS

## 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.


## 9.2 Contributors

Andrey1994 is an author and maintainer of BrainFlow project. Full list of developers who can commit directly to this project and merge Pull Requests can be found here.

- Andrey1994
- daniellasry
- philippitts
- xloem
- matthijscox
- mesca
- retiutut
- trobanga
- shirleyzhang867
- Fan1117
- John42506176Linux
- isaacplotkin
- evaesteban

- purplesyringa
- stellarpower
- Sarthak031
- dvaposto
- myd7349
- alexcastillo
- furmanlukasz
- xgdgsc
- serense
- khoinguyen-hub
- fullflyt
- dependabot[bot]

CHAPTER

TEN

# MIT LICENSE

# MATLAB MODULE INDEX

## b

# INDEX

## A



## B

## C



## D



## E

## F



## G



## I



## L



## M

## N



## P

## R



## S



## T



## W