NxBCI API Documentation

This document provides a comprehensive overview of the API for interacting with the EEG/EMG Device, including various modules and their functionalities.

`MQTT_Receiver`

The `MQTT_Receiver` module is designed to work with an MQTT broker. It subscribes to a specific topic to receive raw EEG data.

Example Usage

```
receiver = MQTT_Receiver(
    channels=16,          # Number of EEG channels
    sample_rate=500,      # Sampling rate in Hz
    duration=4,           # Store latest 4 seconds of EEG data
    ip="192.168.4.2",     # Device IP address
    port=1883,
    topic="esp32-pub-message"
)
data = receiver.eeg_Get_data()  # Returns 16x2000 (channels*sample_rate*duration) deque
receiver.eeg_Stop()
```

Methods

`def eeg_Get_data()`

- **Description**:Return the latest 4 seconds of 16-channel raw EEG data
- **Return**:
  - A `deque` object containing the EEG data.

`def eeg_Stop()`

- **Description**:Shuts down the MQTT receiver.

`def pose_Config(Sample_rate)`

- **Description**:Sets the sampling rate for pose and temperature.
- **Parameters**:
  - `sample_rate` : `float` - The sampling rate for pose and temperature.

`def pose_GetData()`

- **Description**:Retrieves the latest pose and temperature information.
- **Return**: A `deque` containing the lastest pose data:
  - X_a : Acceleration along the x-axis.
  - Y_a : Acceleration along the y-axis.
  - Z_a : Acceleration along the z-axis.
  - T : Temperature in degrees Celsius.
  - ω_x : Angular velocity along the x-axis.
  - ω_y : Angular velocity along the y-axis.
  - ω_z : Angular velocity along the z-axis.

`TCP_Receiver`

The `TCP_Receiver` module facilitates direct data transfer between two devices.

Example Usage

```
receiver = TCP_Receiver(
    channels=16,          # Number of EEG channels
    sample_rate=500,      # Sampling rate in Hz
    duration=4,           # Store latest 4 seconds of EEG data
    ip="192.168.4.2",     # Device IP address
    port=5000,            # Device port
)
data = receiver.eeg_Get_data()  # Returns 16x2000 (channels*sample_rate*duration) deque
receiver.eeg_Stop()
```

Methods

`def eeg_Get_data()`

- **Description**: Retrieves the latest 4 seconds of 16-channel raw EEG data.
- **Return**:
  - A `deque` object containing the EEG data.

`def eeg_Stop()`

- **Description**:Shut down the TCP receiver

`def pose_Config(Sample_rate)`

- **Description**:Set the sampling rate for pose and temperature
- **Parameters**:
  - `sample_rate` : `float` The sampling rate for pose and temperature.

`def pose_GetData()`

- **Description**:Retrieves the latest pose and temperature information.
- **Return**: A `deque` containing the lastest pose data:
  - X_a : Acceleration along the x-axis.
  - Y_a : Acceleration along the y-axis.
  - Z_a : Acceleration along the z-axis.
  - T : Temperature in degrees Celsius.
  - ω_x : Angular velocity along the x-axis.
  - ω_y : Angular velocity along the y-axis.
  - ω_z : Angular velocity along the z-axis.

`Relay_EMQX`

The `Relay_EMQX` module relays raw EEG data to an EMQX cloud server, allowing other devices to receive the same data.

`Example Usage`

```
    relay_emqx = Relay(
    cloud_broker_address='your.cloud.address',  # EMQX cloud broker address
    cloud_port=1883,                            # EMQX cloud port
    cloud_topic="cloud-topic",                  # Target topic on cloud
    client_id='client-id',                      # MQTT client ID
    username='mqtt-user',                       # Authentication username
    password='mqtt-password'                    # Authentication password
)

    receiver.setRelay(relay_emqx)
```

Methods

`def relay_data(data)`

- **Description**:Relays data to the cloud server.
- **Parameters**: `data` : `str` - A string

`def relay_setState(state)`

- **Description**:Enable/disable data Relaying
- **Parameters**: `state` : `bool` - `True` to enable Relaying, `False` to disable

`BluetoothController`

The `BluetoothController` module manages device connections and retrieves device status information.

`Example Usage`

```
    BLE_controller = BluetoothController(
    bluetoothTarget='BLE_FOR_EEG'  # Target is BLE device name
)
    async def main():
        try:
            await BLE_controller.initialize()
            print("JSON File:", BLE_controller.json_data)
        except Exception as e:
            print(f"Error during initialization: {e}")

    if __name__ == "__main__":
            asyncio.run(main())
```

Methods

`async def bt_SetBluetoothTarget(target)`

- **Description**:Sets a new Bluetooth target and attempts connection.
- **Parameters**:
  - `target` : `str` - Device name of BLE device

`async def bt_ReconnectBluetooth()`

- **Description**:Attempts Bluetooth reconnection

`def bt_GetConnectionStatus()`

- **Description**: Checks Bluetooth connection status
- **Return**:
  - `bool` - `True` if connected

`def bt_GetDeviceName()`

- **Description**:Retrieves connected device name
- **Return**:
  - `str` - Device name

`def bt_GetDeviceAddress()`

- **Description**:Retrieves the device MAC address.
- **Return**:
  - `str` - MAC address

`def bt_GetBatteryLevel()`

- **Description**:Checks the battery level.
- **Return**:
  - `int` - Battery percentage (0-100)

`async def bt_SetGain(gain)`

- **Description**: Sets the EEG signal gain.
- **Parameters**:
  - `gain` : `int` - Must be `100` or `1000` (fixed options)

`def bt_GetGain()`

- **Description**:Gets the current gain of the device.
- **Return**:
  - `int` - Current gain value

`async def bt_SetTFcardStorageMode()`

- **Description**:After setting the TF card storage mode, the device will reboot and store the next data to the TF card of the device.

`async def bt_SetTCPMode(wifi_name,password)`

- **Description**:After setting the TCP model, the device will reboot and start the WIFI, and the data can be received after connecting to the WIFI.
- **Return**:
  - `wifi_name` : `str` - The name of the WIFI that the device started
  - `password` : `str` - The password of the WIFI

`async def bt_SetMQTTMode(MQTT_URI,port)`

- **Description**:After setting the MQTT mode, the device will restart and start the WIFI, and after connecting to the WIFI, the MQTT broker service can be started on the user device (PC) to create a message broadcast, and other devices can subscribe to the specified service of the broker.
- **Parameters**:
  - `MQTT_URI` : `str` - The address of the MQTT broker
  - `port` : `int` - The port of the MQTT broker

`Replay`

The `Replay` module reads and replays EEG data from TF card storage.

`Example Usage`

```
    replay = Replay(
    FilePath='path/to/eeg/file'  # TF card file path
)
```

Methods

`def tf_ReadRawData(file_path)`

- **Description**: Loads an EEG file for playback.
- **Parameters**:
  - `file_path` : `str` - File path on TF card

`def tf_GetRawData()`

- **Description**:Retrieves the loaded EEG data.
- **Return**:
  - `dict` -Channel data arrays(e.g., `{'ch1': [...], 'ch2': [...]}`)

`def tf_Seek(file_path,pos)`

- **Description**: Sets playback position of the input file
- **Parameters**:
  - `file_path` : `str` - File that needs to be relocated.
  - `pos` : `int` - Target time domain index

`def tf_SeekSpecifiedSegment(file_path,start_index,end_index)`

- **Description**:Plays a specific data segment.
- **Parameters**:
  - `file_path` : `str` - File that needs to be relocated.
  - `start_index` : `int` - Start index
  - `end_index` : `int` - End index

`def tf_PlayBack(file_path)`

- **Description**:Resets playback to the initial position.
- **Parameters**:
  - `file_path` : `str` - The path of the file that needs to be played back.

`def tf_getDataLength()`

- **Description**:Get the length of the current EEG data
- **Return**:
  - `int` -The length of the current EEG data