Table of Contents

1 Welcome to eConEXG documentation page.

eConEXG is a Python SDK for hardwares made by Niantong Intelligence.

1.1 Installation

`pip install econexg`

Optional dependencies can be installed via `pip install econexg[option1,option2,...]`

```
Available options:
    bdf: save to BDF file functionality support.
    lsl: LSL stream functionality support.
    wifi: iRecorder W32 wifi model support, msvc build tools required.
```

1.2 Quick Start

Example code can be found in examples.

1.3 Platform Support

Hardware	Windows	macOS	Linux
iRecorder USB	✓	✓	✓
iRecorder W8|16	✓	×	×
iRecorder W32	✓	✓	✓
iFocus	✓	✓	✓
TriggerBox	✓	✓	✓

1.4 More

- Discussion Forum, to discuss the project on GitHub
- Issue Tracker, if you run into bugs or have suggestions

I. Devices

2 iRecorder

2.1 `iRecorder`

Bases: `Thread`

Source code in `src\eConEXG\iRecorder\device.py`

```python
class iRecorder(Thread):
    class Dev(Enum):
        SIGNAL = 10  # signal transmission mode
        SIGNAL_START = 11
        IMPEDANCE = 20  # impedance transmission mode
        IMPEDANCE_START = 21
        IDLE = 30  # idle mode
        IDLE_START = 31
        TERMINATE = 40  # Init state
        TERMINATE_START = 41

    def __init__(
        self,
        dev_type: str
    ):
        """
        Args:
            dev_type: iRecorder device type. available options: Literal["W8", "USB8", "W16",
"USB16", "W32", "USB32"]
        Raises:
            Exception: if device type not supported.
            Exception: if adapter not available.
        """
        if dev_type not in {"W8", "USB8", "W16", "USB16", "W32", "USB32"}:
            raise ValueError("Unsupported device type.")
        super().__init__(daemon=True, name=f"iRecorder {dev_type}")
        self.handler = None
        self.__info_q = Queue(128)
        self.__with_q = True
        self.__error_message = "Device not connected, please connect first."
        self.__save_data = Queue()
        self.__status = iRecorder.Dev.TERMINATE
        self.__lsl_flag = False
        self.__bdf_flag = False
        self.__dev_args = {"type": dev_type}
        self.__dev_args.update({"channel": self.__get_chs()})

        self.__parser = Parser(self.__dev_args["channel"])
        self.__interface = get_interface(dev_type)(self.__info_q)
        self.__dev_sock = get_sock(dev_type)
        self.__dev_args.update({"AdapterInfo": self.__interface.interface})

        self._bdf_file = None
        self.dev = None

        self.set_frequency()
        self.update_channels()

    def find_devs(self, duration: Optional[int] = None) -> Optional[list]:
        """
        Search for available devices, can only be called once per instance.

        Args:
            duration: Search interval in seconds, blocks for about `duration` seconds and
return found devices,
                if set to `None`, return `None` immediately, devices can later be acquired by
calling `get_devs()` in a loop.

        Returns:
            Available devices.

        Raises:
```

```
75              Exception: If search thread already running or iRecorder already connected.
76          """
77          if self.is_alive():
78              raise Exception("iRecorder already connected.")
79          if self.__interface.is_alive():
80              raise Exception("Search thread already running.")
81          self.__interface.start()
82          if duration is None:
83              return
84          start = time.time()
85          while time.time() - start < duration:
86              time.sleep(0.5)
87          self.__finish_search()
88          return self.get_devs()
89
90      def get_devs(self, verbose: bool = False) -> list:
91          """
92          Get available devices. This can be called after `find_devs(duration = None)` in a
93  loop,
94              each call will *only* return newly found devices.
95
96          Args:
97              verbose: if True, return all available devices information, otherwise only return
98  names for connection,
99                  if you don't know what this parameter does, just leave it at its default
100 value.
101
102         Returns:
103             Newly found devices.
104
105         Raises:
106             Exception: adapter not found or not enabled etc.
107         """
108         ret = []
109         time.sleep(0.1)
110         while not self.__info_q.empty():
111             info = self.__info_q.get()
112             if isinstance(info, list):
113                 ret.append(info if verbose else info[-1])
114             elif isinstance(info, bool):
115                 if verbose:
116                     ret.append(info)
117             elif isinstance(info, str):
118                 raise Exception(info)
119         return ret
120
121     def get_dev_status(self) -> str:
122         """
123         Get current device status.
124
125         Returns:
126             "SIGNAL": data acquisition mode
127             "IMPEDANCE": impedance acquisition mode
128             "IDLE": idle mode
129             "TERMINATE": device not connected or connection closed.
130         """
131         return self.__status.name
132
133     def get_dev_info(self) -> dict:
134         """
135         Get current device information, including device name, hardware channel number,
136 acquired channels, sample frequency, etc.
137
```

```
138         Returns:
139             A dictionary containing device information, which includes:
140                 `type`: hardware type;
141                 `channel`: hardware channel number;
142                 `AdapterInfo`: adapter used for connection;
143                 `fs`: sample frequency in Hz;
144                 `ch_info`: channel dictionary, including channel index and name, can be
145 altered by `update_channels()`.
146         """
147         return deepcopy(self.__dev_args)
148 
149     @staticmethod
150     def get_available_frequency(dev_type: str) -> list:
151         """Get available sample frequencies of different device types.
152 
153         Returns:
154             Available sample frequencies in Hz.
155         """
156         if "USB" in dev_type:
157             return [500, 1000, 2000]
158         return [500]
159 
160     def set_frequency(self, fs: Optional[int] = None):
161         """Update device sample frequency, this method should be invoked before
162 `connect_device`.
163 
164         Args:
165             fs: sample frequency in Hz, if `fs` is set to `None` or not in
166 `get_available_frequency()`,
167                 it will fall back to the lowest available frequency.
168 
169         Raises:
170             Exception: Device is already connected.
171 
172         New in:
173             - now you can set the sample frequency after device connection.
174         """
175         if self.__status not in [iRecorder.Dev.IDLE, iRecorder.Dev.TERMINATE]:
176             warn = "Device acquisition in progress, please `stop_acquisition()` first."
177             raise Exception(warn)
178         available = self.get_available_frequency(self.__dev_args["type"])
179         default = available[0]
180         if fs is None:
181             fs = default
182         if fs not in available:
183             print(f"Invalid sample frequency, fallback to {default}Hz")
184             fs = default
185         self.__dev_args.update({"fs": fs})
186         self.__parser._update_fs(fs)
187         if self.dev is not None:
188             self.dev.set_fs(fs)
189 
190     def connect_device(self, addr: str) -> None:
191         """
192         Connect to device by address, block until connection is established or failed.
193 
194         Args:
195             addr: device address.
196 
197         Raises:
198             Exception: if device already connected or connection establishment failed.
199         """
200         if self.is_alive():
```

```
            raise Exception("iRecorder already connected")
        try:
            ret = self.__interface.connect(addr)
            self.__dev_args.update({"name": addr, "sock": ret})
            self.__dev_args.update({"_length": self.__parser.packet_len})
            self.dev = self.__dev_sock(self.__dev_args)
            self.__parser.batt_val = self.dev.send_heartbeat()
            self.__error_message = None
            self.__status = iRecorder.Dev.IDLE_START
            self.start()
        except Exception as e:
            self.__error_message = "Device connection failed."
            self.__finish_search()
            raise e

    def update_channels(self, channels: Optional[dict] = None):
        """
        Update channels to acquire, invoke this method when device is not acquiring data or
impedance.

        Args:
            channels: channel number and name mapping, e.g. `{0: "FPz", 1: "Oz", 2: "CPz"}`,
                if `None` is given, reset to all available channels with default names.

        Raises:
            Exception: if data/impedance acquisition in progress.
        """
        if self.__status not in [iRecorder.Dev.IDLE, iRecorder.Dev.TERMINATE]:
            warn = "Device acquisition in progress, please stop_acquisition() first."
            raise Exception(warn)
        if channels is None:
            from .default_config import getChannels

            channels = getChannels(self.__dev_args["channel"])
        self.__dev_args.update({"ch_info": channels})
        ch_idx = [i for i in channels.keys()]
        self.__parser._update_chs(ch_idx)

    def start_acquisition_data(self, with_q: bool = True):
        """
        Send data acquisition command to device, block until data acquisition started or
failed.

        Args:
            with_q: if True, signal data will be stored in a queue and **should** be acquired
by calling `get_data()` in a loop in case data queue is full.
                if False, new data will not be directly available and can only be acquired
through `open_lsl_stream` and `save_bdf_file`.

        Raises:
            Exception: if device not connected or data acquisition init failed.
        """
        self.__check_dev_status()
        self.__with_q = with_q
        if self.__status == iRecorder.Dev.SIGNAL:
            return
        if self.__status == iRecorder.Dev.IMPEDANCE:
            self.stop_acquisition()
        self.__status = iRecorder.Dev.SIGNAL_START
        while self.__status not in [iRecorder.Dev.SIGNAL, iRecorder.Dev.TERMINATE]:
            time.sleep(0.01)
        self.__check_dev_status()

```

```
    def get_data(
        self, timeout: Optional[float] = 0.02
    ) -> Optional[list[Optional[list]]]:
        """
        Acquire all available data, make sure this function is called in a loop when `with_q`
is set to `True` in`start_acquisition_data()`

        Args:
            timeout: Non-negative value, blocks at most `timeout` seconds and return, if set
to `None`, blocks until new data is available.

        Returns:
            A list of frames, each frame is a list contains all wanted eeg channels and
trigger box channel,
                eeg channels can be updated by `update_channels()`.

        Data Unit:
            - eeg: micro volts (μV)
            - triggerbox: int, from `0` to `255`

        Raises:
            Exception: if device not connected or in data acquisition mode.
        """
        self.__check_dev_status()
        if not self.__with_q:
            return
        if self.__status != iRecorder.Dev.SIGNAL:
            raise Exception("Data acquisition not started, please start first.")
        try:
            data: list = self.__save_data.get(timeout=timeout)
        except queue.Empty:
            return []
        while not self.__save_data.empty():
            data.extend(self.__save_data.get())
        return data

    def stop_acquisition(self) -> None:
        """
        Stop data or impedance acquisition, block until data acquisition stopped or failed.

        Raises:
            Exception: if device not connected or acquisition stop failed.
        """
        self.__check_dev_status()
        if self.__status == iRecorder.Dev.IDLE:
            return
        self.__status = iRecorder.Dev.IDLE_START
        while self.__status not in [iRecorder.Dev.IDLE, iRecorder.Dev.TERMINATE]:
            time.sleep(0.01)
        self.__check_dev_status()

    def start_acquisition_impedance(self) -> None:
        """
        Send impedance acquisition command to device, block until data acquisition started or
failed.

        Raises:
            Exception: if device not connected or impedance acquisition init failed.
        """
        self.__check_dev_status()
        if self.__status == iRecorder.Dev.IMPEDANCE:
            return
        if self.__status == iRecorder.Dev.SIGNAL:
```

```
            self.stop_acquisition()
        self.__status = iRecorder.Dev.IMPEDANCE_START
        while self.__status not in [iRecorder.Dev.IMPEDANCE, iRecorder.Dev.TERMINATE]:
            time.sleep(0.01)
        self.__check_dev_status()

    def get_impedance(self) -> Optional[list]:
        """
        Acquire channel impedances, return immediately, impedance update interval is about
2000ms.

        Returns:
            A list of channel impedance ranging from `0` to `np.inf` if available, otherwise
`None`.

        Data Unit:
            - impedance: ohm (Ω)
        """
        self.__check_dev_status()
        return self.__parser.impedance

    def close_dev(self) -> None:
        """
        Close device connection and release resources, resources are automatically released
on device error.
        """
        if self.__status != iRecorder.Dev.TERMINATE:
            # ensure socket is closed correctly
            self.__status = iRecorder.Dev.TERMINATE_START
            while self.__status != iRecorder.Dev.TERMINATE:
                time.sleep(0.01)
        if self.is_alive():
            self.join()

    def get_packet_drop_times(self) -> int:
        """
        Retrieve packet drop times.
        This value accumulates during data transmission and will be reset to `0` after device
status change.

        Returns:
            accumulated packet drop times.
        """
        return self.__parser._drop_count

    def get_battery_value(self) -> int:
        """
        Query battery level.

        Returns:
            battery level in percentage, range from `0` to `100`.
        """
        return self.__parser.batt_val

    def open_lsl_stream(self):
        """
        Open LSL stream, can be invoked after `start_acquisition_data()`,
            each frame is the same as described in `get_data()`.

        Raises:
            Exception: if data acquisition not started or LSL stream already opened.
            LSLException: if LSL stream creation failed.
            importError: if `pylsl` not installed or liblsl not installed on unix like
```

```
system.
        """
        if self.__status != iRecorder.Dev.SIGNAL:
            raise Exception("Data acquisition not started, please start first.")
        if hasattr(self, "_lsl_stream"):
            raise Exception("LSL stream already opened.")
        from ..utils.lslWrapper import lslSender

        self._lsl_stream = lslSender(
            self.__dev_args["ch_info"],
            f"iRe{self.__dev_args['type']}_{self.__dev_args['name'][-2:]}",
            "EEG",
            self.__dev_args["fs"],
            with_trigger=True,
        )
        self.__lsl_flag = True

    def close_lsl_stream(self):
        """
        Close LSL stream manually, invoked automatically after `stop_acquisition()` or
`close_dev()`
        """
        self.__lsl_flag = False
        if hasattr(self, "_lsl_stream"):
            del self._lsl_stream

    def create_bdf_file(self, filename: str):
        """
        Create a BDF file and save data to it, invoke it after `start_acquisition_data()`.

        Args:
            filename: file name to save data, accept absolute or relative path.

        Raises:
            Exception: if data acquisition not started or `save_bdf_file` is invoked and BDF
file already created.
            OSError: if BDF file creation failed, this may be caused by invalid file path or
permission issue.
            importError: if `pyedflib` is not installed.
        """
        if self.__status != iRecorder.Dev.SIGNAL:
            raise Exception("Data acquisition not started")
        if self._bdf_file is not None:
            raise Exception("BDF file already created.")
        from ..utils.bdfWrapper import bdfSaverIRecorder

        if filename[-4:].lower() != ".bdf":
            filename += ".bdf"
        self._bdf_file = bdfSaverIRecorder(
            filename,
            self.__dev_args["ch_info"],
            self.__dev_args["fs"],
            f"iRecorder_{self.__dev_args['type']}_{self.__dev_args['name']}",
        )
        self.__bdf_flag = True

    def close_bdf_file(self):
        """
        Close and save BDF file manually, invoked automatically after `stop_acquisition()` or
`close_dev()`
        """
        self.__bdf_flag = False
        if self._bdf_file is not None:
```

```
453             self._bdf_file.close_bdf()
454             self._bdf_file = None
455 
456     def send_bdf_marker(self, marker: str):
457         """
458         Send marker to BDF file, can be invoked after `create_bdf_file()`, otherwise it will
459 be ignored.
460 
461         Args:
462             marker: marker string to write.
463         """
464         if self._bdf_file is not None:
465             self._bdf_file.write_Annotation(marker)
466 
467     # def set_callback_handler(self, handler: Callable[[Optional[str]], None]):
468     #     """
469     #     Set callback handler function, invoked automatically when device thread ended if
470 set.
471 
472     #     Args:
473     #         handler: a callable function that takes a string of error message or `None` as
474 input.
475     #     """
476     #     self.handler = handler
477 
478     def __check_dev_status(self):
479         if self.__error_message is None:
480             return
481         if self.is_alive():
482             self.close_dev()
483         raise Exception(self.__error_message)
484 
485     def run(self):
486         while self.__status not in [iRecorder.Dev.TERMINATE_START]:
487             if self.__status == iRecorder.Dev.SIGNAL_START:
488                 self.__recv_data(imp_mode=False)
489             elif self.__status == iRecorder.Dev.IMPEDANCE_START:
490                 self.__recv_data(imp_mode=True)
491             elif self.__status in [iRecorder.Dev.IDLE_START]:
492                 self.__idle_state()
493             else:
494                 self.__error_message = f"Unknown status: {self.__status}"
495                 break
496         try:
497             self.dev.close_socket()
498         except Exception:
499             pass
500         finally:
501             self.__finish_search()
502             self.__status = iRecorder.Dev.TERMINATE
503         # if self.handler is not None:
504         #     self.handler(self.__error_message)
505 
506     def __recv_data(self, imp_mode=True):
507         self.__parser.imp_flag = imp_mode
508         retry = 0
509         try:
510             if imp_mode:
511                 self.dev.start_impe()
512                 self.__status = iRecorder.Dev.IMPEDANCE
513                 print("IMPEDANCE START")
514             else:
515                 self.dev.start_data()
```

```
                self.__status = iRecorder.Dev.SIGNAL
                print("SIGNAL START")
            except Exception:
                self.__error_message = "Data/Impedance mode initialization failed."
                self.__status = iRecorder.Dev.TERMINATE_START
            # recv data
            while self.__status in [iRecorder.Dev.SIGNAL, iRecorder.Dev.IMPEDANCE]:
                try:
                    data = self.dev.recv_socket()
                    if not data:
                        raise Exception("Remote end closed.")
                    ret = self.__parser.parse_data(data)
                    if ret:
                        if self.__with_q:
                            self.__save_data.put(ret)
                        if self.__bdf_flag:
                            self._bdf_file.write_chunk(ret)
                        if self.__lsl_flag:
                            self._lsl_stream.push_chunk(ret)
                except Exception:
                    if (self.__dev_args["type"] == "W32") and (retry < 1):
                        try:
                            print("Wi-Fi reconnecting...")
                            self.dev.close_socket()
                            self.dev = self.__dev_sock(self.__dev_args, retry_timeout=3)
                            retry += 1
                            continue
                        except Exception:
                            print("Wi-Fi reconnection failed")
                    self.__error_message = "Data transmission timeout."
                    self.__status = iRecorder.Dev.TERMINATE_START
            # postprocess
            self.close_bdf_file()
            self.close_lsl_stream()
            self.__parser.clear_buffer()
            while not self.__save_data.empty():
                self.__save_data.get()
            # stop recv data
            if self.__status != iRecorder.Dev.TERMINATE_START:
                try:  # stop data acquisition when thread ended
                    self.dev.stop_recv()
                except Exception:
                    if self.__status == iRecorder.Dev.IDLE_START:
                        self.__error_message = "Device connection lost."
                    self.__status = iRecorder.Dev.TERMINATE_START

        def __idle_state(self):
            timestamp = time.time()
            self.__status = iRecorder.Dev.IDLE
            while self.__status in [iRecorder.Dev.IDLE]:
                if (time.time() - timestamp) < 5:
                    time.sleep(0.2)  # to reduce cpu usage
                    continue
                try:  # heartbeat to keep socket alive and update battery level
                    self.__parser.batt_val = self.dev.send_heartbeat()
                    timestamp = time.time()
                    # print("Ah, ah, ah, ah\nStayin' alive, stayin' alive")
                except Exception:
                    self.__error_message = "Device connection lost."
                    self.__status = iRecorder.Dev.TERMINATE_START

        def __get_chs(self) -> int:
            return int("".join([i for i in self.__dev_args["type"] if i.isdigit()]))
```

```
        def __finish_search(self):
            if self.__interface.is_alive():
                self.__interface.stop()
                self.__interface.join()
            while not self.__info_q.empty():
                self.__info_q.get()
```

2.1.1 `__init__(dev_type)`

Parameters:

Name	Type	Description	Default
`dev_type`	`str`	iRecorder device type. available options: Literal["W8", "USB8", "W16", "USB16", "W32", "USB32"]	*required*

Raises: Exception: if device type not supported. Exception: if adapter not available.

Source code in `src\eConEXG\iRecorder\device.py`

```
24  def __init__(
25      self,
26      dev_type: str
27  ):
28      """
29      Args:
30          dev_type: iRecorder device type. available options: Literal["W8", "USB8", "W16",
31  "USB16", "W32", "USB32"]
32      Raises:
33          Exception: if device type not supported.
34          Exception: if adapter not available.
35      """
36      if dev_type not in {"W8", "USB8", "W16", "USB16", "W32", "USB32"}:
37          raise ValueError("Unsupported device type.")
38      super().__init__(daemon=True, name=f"iRecorder {dev_type}")
39      self.handler = None
40      self.__info_q = Queue(128)
41      self.__with_q = True
42      self.__error_message = "Device not connected, please connect first."
43      self.__save_data = Queue()
44      self.__status = iRecorder.Dev.TERMINATE
45      self.__lsl_flag = False
46      self.__bdf_flag = False
47      self.__dev_args = {"type": dev_type}
48      self.__dev_args.update({"channel": self.__get_chs()})
49
50      self.__parser = Parser(self.__dev_args["channel"])
51      self.__interface = get_interface(dev_type)(self.__info_q)
52      self.__dev_sock = get_sock(dev_type)
53      self.__dev_args.update({"AdapterInfo": self.__interface.interface})
54
55      self._bdf_file = None
56      self.dev = None
57
58      self.set_frequency()
        self.update_channels()
```

2.1.2 `close_bdf_file()`

Close and save BDF file manually, invoked automatically after `stop_acquisition()` or `close_dev()`

Source code in `src\eConEXG\iRecorder\device.py`

```
420 def close_bdf_file(self):
421     """
422     Close and save BDF file manually, invoked automatically after `stop_acquisition()` or
423 `close_dev()`
424     """
425     self.__bdf_flag = False
426     if self._bdf_file is not None:
427         self._bdf_file.close_bdf()
            self._bdf_file = None
```

2.1.3 `close_dev()`

Close device connection and release resources, resources are automatically released on device error.

Source code in `src\eConEXG\iRecorder\device.py`

```
328  def close_dev(self) -> None:
329      """
330      Close device connection and release resources, resources are automatically released on
331  device error.
332      """
333      if self.__status != iRecorder.Dev.TERMINATE:
334          # ensure socket is closed correctly
335          self.__status = iRecorder.Dev.TERMINATE_START
336          while self.__status != iRecorder.Dev.TERMINATE:
337              time.sleep(0.01)
338      if self.is_alive():
             self.join()
```

2.1.4 `close_lsl_stream()`

Close LSL stream manually, invoked automatically after `stop_acquisition()` or `close_dev()`

Source code in `src\eConEXG\iRecorder\device.py`

```
384  def close_lsl_stream(self):
385      """
386      Close LSL stream manually, invoked automatically after `stop_acquisition()` or
387  `close_dev()`
388      """
389      self.__lsl_flag = False
390      if hasattr(self, "_lsl_stream"):
             del self._lsl_stream
```

2.1.5 `connect_device(addr)`

Connect to device by address, block until connection is established or failed.

Parameters:

Name	Type	Description	Default
`addr`	`str`	device address.	*required*

Raises:

Type	Description
`Exception`	if device already connected or connection establishment failed.

Source code in `src\eConEXG\iRecorder\device.py`

```
180  def connect_device(self, addr: str) -> None:
181      """
182      Connect to device by address, block until connection is established or failed.
183
184      Args:
185          addr: device address.
186
187      Raises:
188          Exception: if device already connected or connection establishment failed.
189      """
190      if self.is_alive():
191          raise Exception("iRecorder already connected")
192      try:
193          ret = self.__interface.connect(addr)
194          self.__dev_args.update({"name": addr, "sock": ret})
195          self.__dev_args.update({"_length": self.__parser.packet_len})
196          self.dev = self.__dev_sock(self.__dev_args)
197          self.__parser.batt_val = self.dev.send_heartbeat()
198          self.__error_message = None
199          self.__status = iRecorder.Dev.IDLE_START
200          self.start()
201      except Exception as e:
202          self.__error_message = "Device connection failed."
203          self.__finish_search()
204          raise e
```

2.1.6 `create_bdf_file(filename)`

Create a BDF file and save data to it, invoke it after `start_acquisition_data()`.

Parameters:

Name	Type	Description	Default
`filename`	`str`	file name to save data, accept absolute or relative path.	*required*

Raises:

Type	Description
`Exception`	if data acquisition not started or `save_bdf_file` is invoked and BDF file already created.

Type	Description
`OSError`	if BDF file creation failed, this may be caused by invalid file path or permission issue.
`importError`	if `pyedflib` is not installed.

Source code in `src\eConEXG\iRecorder\device.py`

```
392     def create_bdf_file(self, filename: str):
393         """
394         Create a BDF file and save data to it, invoke it after `start_acquisition_data()`.
395 
396         Args:
397             filename: file name to save data, accept absolute or relative path.
398 
399         Raises:
400             Exception: if data acquisition not started or `save_bdf_file` is invoked and BDF file
401     already created.
402             OSError: if BDF file creation failed, this may be caused by invalid file path or
403     permission issue.
404             importError: if `pyedflib` is not installed.
405         """
406         if self.__status != iRecorder.Dev.SIGNAL:
407             raise Exception("Data acquisition not started")
408         if self._bdf_file is not None:
409             raise Exception("BDF file already created.")
410         from ..utils.bdfWrapper import bdfSaverIRecorder
411 
412         if filename[-4:].lower() != ".bdf":
413             filename += ".bdf"
414         self._bdf_file = bdfSaverIRecorder(
415             filename,
416             self.__dev_args["ch_info"],
417             self.__dev_args["fs"],
418             f"iRecorder_{self.__dev_args['type']}_{self.__dev_args['name']}",
        )
            self.__bdf_flag = True
```

2.1.7 `find_devs(duration=None)`

Search for available devices, can only be called once per instance.

Parameters:

Name	Type	Description	Default
`duration`	`Optional[int]`	Search interval in seconds, blocks for about `duration` seconds and return found devices, if set to `None`, return	`None`

Name	Type	Description	Default
		`None` immediately, devices can later be acquired by calling `get_devs()` in a loop.	

Returns:

Type	Description
`Optional[list]`	Available devices.

Raises:

Type	Description
`Exception`	If search thread already running or iRecorder already connected.

Source code in `src\eConEXG\iRecorder\device.py`

```
60  def find_devs(self, duration: Optional[int] = None) -> Optional[list]:
61      """
62      Search for available devices, can only be called once per instance.
63
64      Args:
65          duration: Search interval in seconds, blocks for about `duration` seconds and return
66  found devices,
67              if set to `None`, return `None` immediately, devices can later be acquired by
68  calling `get_devs()` in a loop.
69
70      Returns:
71          Available devices.
72
73      Raises:
74          Exception: If search thread already running or iRecorder already connected.
75      """
76      if self.is_alive():
77          raise Exception("iRecorder already connected.")
78      if self.__interface.is_alive():
79          raise Exception("Search thread already running.")
80      self.__interface.start()
81      if duration is None:
82          return
83      start = time.time()
84      while time.time() - start < duration:
85          time.sleep(0.5)
        self.__finish_search()
        return self.get_devs()
```

2.1.8 `get_available_frequency(dev_type)` `staticmethod`

Get available sample frequencies of different device types.

Returns:

Type	Description
`list`	Available sample frequencies in Hz.

Source code in `src\eConEXG\iRecorder\device.py`

```
141     @staticmethod
142     def get_available_frequency(dev_type: str) -> list:
143         """Get available sample frequencies of different device types.
144 
145         Returns:
146             Available sample frequencies in Hz.
147         """
148         if "USB" in dev_type:
149             return [500, 1000, 2000]
150         return [500]
```

2.1.9 `get_battery_value()`

Query battery level.

Returns:

Type	Description
`int`	battery level in percentage, range from `0` to `100`.

Source code in `src\eConEXG\iRecorder\device.py`

```
350     def get_battery_value(self) -> int:
351         """
352         Query battery level.
353 
354         Returns:
355             battery level in percentage, range from `0` to `100`.
356         """
357         return self.__parser.batt_val
```

2.1.10 `get_data(timeout=0.02)`

Acquire all available data, make sure this function is called in a loop when `with_q` is set to `True` in `start_acquisition_data()`

Parameters:

Name	Type	Description	Default
`timeout`	`Optional[float]`	Non-negative value, blocks at most `timeout` seconds and return, if set to `None`, blocks until new data is available.	`0.02`

Returns:

Type	Description
`Optional[list[Optional[list]]]`	A list of frames, each frame is a list contains all wanted eeg channels and trigger box channel, eeg channels can be updated by `update_channels()`.

Data Unit

- eeg: micro volts (μV)
- triggerbox: int, from `0` to `255`

Raises:

Type	Description
`Exception`	if device not connected or in data acquisition mode.

Source code in `src\eConEXG\iRecorder\device.py`

```
250  def get_data(
251      self, timeout: Optional[float] = 0.02
252  ) -> Optional[list[Optional[list]]]:
253      """
254      Acquire all available data, make sure this function is called in a loop when `with_q` is
255  set to `True` in`start_acquisition_data()`
256
257      Args:
258          timeout: Non-negative value, blocks at most `timeout` seconds and return, if set to
259  `None`, blocks until new data is available.
260
261      Returns:
262          A list of frames, each frame is a list contains all wanted eeg channels and trigger
263  box channel,
264              eeg channels can be updated by `update_channels()`.
265
266      Data Unit:
267          - eeg: micro volts (μV)
268          - triggerbox: int, from `0` to `255`
269
270      Raises:
271          Exception: if device not connected or in data acquisition mode.
272      """
273      self.__check_dev_status()
274      if not self.__with_q:
275          return
276      if self.__status != iRecorder.Dev.SIGNAL:
277          raise Exception("Data acquisition not started, please start first.")
278      try:
279          data: list = self.__save_data.get(timeout=timeout)
280      except queue.Empty:
281          return []
          while not self.__save_data.empty():
              data.extend(self.__save_data.get())
          return data
```

2.1.11 `get_dev_info()`

Get current device information, including device name, hardware channel number, acquired channels, sample frequency, etc.

Returns:

Type	Description
`dict`	A dictionary containing device information, which includes: `type`: hardware type; `channel`: hardware channel number; `AdapterInfo`: adapter used for connection; `fs`: sample frequency in Hz; `ch_info`: channel dictionary, including channel index and name, can be altered by `update_channels()`.

Source code in `src\eConEXG\iRecorder\device.py`

```
127  def get_dev_info(self) -> dict:
128      """
129      Get current device information, including device name, hardware channel number, acquired
130  channels, sample frequency, etc.
131
132      Returns:
133          A dictionary containing device information, which includes:
134              `type`: hardware type;
135              `channel`: hardware channel number;
136              `AdapterInfo`: adapter used for connection;
137              `fs`: sample frequency in Hz;
138              `ch_info`: channel dictionary, including channel index and name, can be altered
139  by `update_channels()`.
          """
          return deepcopy(self.__dev_args)
```

2.1.12 `get_dev_status()`

Get current device status.

Returns:

Type	Description
`str`	"SIGNAL": data acquisition mode
`str`	"IMPEDANCE": impedance acquisition mode
`str`	"IDLE": idle mode
`str`	"TERMINATE": device not connected or connection closed.

Source code in `src\eConEXG\iRecorder\device.py`

```
115  def get_dev_status(self) -> str:
116      """
117      Get current device status.
118
119      Returns:
120          "SIGNAL": data acquisition mode
121          "IMPEDANCE": impedance acquisition mode
122          "IDLE": idle mode
123          "TERMINATE": device not connected or connection closed.
124      """
125      return self.__status.name
```

2.1.13 `get_devs(verbose=False)`

Get available devices. This can be called after `find_devs(duration = None)` in a loop, each call will *only* return newly found devices.

Parameters:

Name	Type	Description	Default
`verbose`	`bool`	if True, return all available devices information, otherwise only return names for connection, if you don't know what this parameter does, just leave it at its default value.	`False`

Returns:

Type	Description
`list`	Newly found devices.

Raises:

Type	Description
`Exception`	adapter not found or not enabled etc.

Source code in `src\eConEXG\iRecorder\device.py`

```
87  def get_devs(self, verbose: bool = False) -> list:
88      """
89      Get available devices. This can be called after `find_devs(duration = None)` in a loop,
90          each call will *only* return newly found devices.
91
92      Args:
93          verbose: if True, return all available devices information, otherwise only return
94  names for connection,
95              if you don't know what this parameter does, just leave it at its default value.
96
97      Returns:
98          Newly found devices.
99
100     Raises:
101         Exception: adapter not found or not enabled etc.
102     """
103     ret = []
104     time.sleep(0.1)
105     while not self.__info_q.empty():
106         info = self.__info_q.get()
107         if isinstance(info, list):
108             ret.append(info if verbose else info[-1])
109         elif isinstance(info, bool):
110             if verbose:
111                 ret.append(info)
112         elif isinstance(info, str):
113             raise Exception(info)
        return ret
```

2.1.14 `get_impedance()`

Acquire channel impedances, return immediately, impedance update interval is about 2000ms.

Returns:

Type	Description
`Optional[list]`	A list of channel impedance ranging from `0` to `np.inf` if available, otherwise `None`.

Data Unit

- impedance: ohm (Ω)

Source code in `src\eConEXG\iRecorder\device.py`

```
315  def get_impedance(self) -> Optional[list]:
316      """
317      Acquire channel impedances, return immediately, impedance update interval is about
318  2000ms.
319
320      Returns:
321          A list of channel impedance ranging from `0` to `np.inf` if available, otherwise
322  `None`.
323
324      Data Unit:
325          - impedance: ohm (Ω)
326      """
           self.__check_dev_status()
           return self.__parser.impedance
```

2.1.15 `get_packet_drop_times()`

Retrieve packet drop times. This value accumulates during data transmission and will be reset to `0` after device status change.

Returns:

Type	Description
`int`	accumulated packet drop times.

Source code in `src\eConEXG\iRecorder\device.py`

```
340  def get_packet_drop_times(self) -> int:
341      """
342      Retrieve packet drop times.
343      This value accumulates during data transmission and will be reset to `0` after device
344  status change.
345
346      Returns:
347          accumulated packet drop times.
348      """
           return self.__parser._drop_count
```

2.1.16 `open_lsl_stream()`

Open LSL stream, can be invoked after `start_acquisition_data()`, each frame is the same as described in `get_data()`.

Raises:

Type	Description
`Exception`	if data acquisition not started or LSL stream already opened.
`LSLException`	if LSL stream creation failed.
`importError`	if `pylsl` not installed or liblsl not installed on unix like system.

Source code in `src\eConEXG\iRecorder\device.py`

```
359     def open_lsl_stream(self):
360         """
361         Open LSL stream, can be invoked after `start_acquisition_data()`,
362             each frame is the same as described in `get_data()`.
363 
364         Raises:
365             Exception: if data acquisition not started or LSL stream already opened.
366             LSLException: if LSL stream creation failed.
367             importError: if `pylsl` not installed or liblsl not installed on unix like system.
368         """
369         if self.__status != iRecorder.Dev.SIGNAL:
370             raise Exception("Data acquisition not started, please start first.")
371         if hasattr(self, "_lsl_stream"):
372             raise Exception("LSL stream already opened.")
373         from ..utils.lslWrapper import lslSender
374 
375         self._lsl_stream = lslSender(
376             self.__dev_args["ch_info"],
377             f"iRe{self.__dev_args['type']}_{self.__dev_args['name'][-2:]}",
378             "EEG",
379             self.__dev_args["fs"],
380             with_trigger=True,
381         )
382         self.__lsl_flag = True
```

2.1.17 `send_bdf_marker(marker)`

Send marker to BDF file, can be invoked after `create_bdf_file()`, otherwise it will be ignored.

Parameters:

Name	Type	Description	Default
`marker`	`str`	marker string to write.	*required*

Source code in `src\eConEXG\iRecorder\device.py`

```
429  def send_bdf_marker(self, marker: str):
430      """
431      Send marker to BDF file, can be invoked after `create_bdf_file()`, otherwise it will be
432  ignored.
433
434      Args:
435          marker: marker string to write.
436      """
437      if self._bdf_file is not None:
             self._bdf_file.write_Annotation(marker)
```

2.1.18 `set_frequency(fs=None)`

Update device sample frequency, this method should be invoked before `connect_device`.

Parameters:

Name	Type	Description	Default
`fs`	`Optional[int]`	sample frequency in Hz, if `fs` is set to `None` or not in `get_available_frequency()`, it will fall back to the lowest available frequency.	`None`

Raises:

Type	Description
`Exception`	Device is already connected.

New in

- now you can set the sample frequency after device connection.

Source code in `src\eConEXG\iRecorder\device.py`

```
152  def set_frequency(self, fs: Optional[int] = None):
153      """Update device sample frequency, this method should be invoked before `connect_device`.
154
155      Args:
156          fs: sample frequency in Hz, if `fs` is set to `None` or not in
157  `get_available_frequency()`,
158              it will fall back to the lowest available frequency.
159
160      Raises:
161          Exception: Device is already connected.
162
163      New in:
164          - now you can set the sample frequency after device connection.
165      """
166      if self.__status not in [iRecorder.Dev.IDLE, iRecorder.Dev.TERMINATE]:
167          warn = "Device acquisition in progress, please `stop_acquisition()` first."
168          raise Exception(warn)
169      available = self.get_available_frequency(self.__dev_args["type"])
170      default = available[0]
171      if fs is None:
172          fs = default
173      if fs not in available:
174          print(f"Invalid sample frequency, fallback to {default}Hz")
175          fs = default
176      self.__dev_args.update({"fs": fs})
177      self.__parser._update_fs(fs)
178      if self.dev is not None:
             self.dev.set_fs(fs)
```

2.1.19 `start_acquisition_data(with_q=True)`

Send data acquisition command to device, block until data acquisition started or failed.

Parameters:

Name	Type	Description	Default
`with_q`	`bool`	if True, signal data will be stored in a queue and **should** be acquired by calling `get_data()` in a loop in case data queue is full. if False, new data will not be directly available and can only be acquired through `open_lsl_stream` and `save_bdf_file`.	`True`

Raises:

Type	Description
`Exception`	if device not connected or data acquisition init failed.

Source code in `src\eConEXG\iRecorder\device.py`

```
228  def start_acquisition_data(self, with_q: bool = True):
229      """
230      Send data acquisition command to device, block until data acquisition started or failed.
231
232      Args:
233          with_q: if True, signal data will be stored in a queue and **should** be acquired by
234  calling `get_data()` in a loop in case data queue is full.
235              if False, new data will not be directly available and can only be acquired
236  through `open_lsl_stream` and `save_bdf_file`.
237
238      Raises:
239          Exception: if device not connected or data acquisition init failed.
240      """
241      self.__check_dev_status()
242      self.__with_q = with_q
243      if self.__status == iRecorder.Dev.SIGNAL:
244          return
245      if self.__status == iRecorder.Dev.IMPEDANCE:
246          self.stop_acquisition()
247      self.__status = iRecorder.Dev.SIGNAL_START
248      while self.__status not in [iRecorder.Dev.SIGNAL, iRecorder.Dev.TERMINATE]:
          time.sleep(0.01)
      self.__check_dev_status()
```

2.1.20 `start_acquisition_impedance()`

Send impedance acquisition command to device, block until data acquisition started or failed.

Raises:

Type	Description
`Exception`	if device not connected or impedance acquisition init failed.

Source code in `src\eConEXG\iRecorder\device.py`

```
298  def start_acquisition_impedance(self) -> None:
299      """
300      Send impedance acquisition command to device, block until data acquisition started or
301  failed.
302
303      Raises:
304          Exception: if device not connected or impedance acquisition init failed.
305      """
306      self.__check_dev_status()
307      if self.__status == iRecorder.Dev.IMPEDANCE:
308          return
309      if self.__status == iRecorder.Dev.SIGNAL:
310          self.stop_acquisition()
311      self.__status = iRecorder.Dev.IMPEDANCE_START
312      while self.__status not in [iRecorder.Dev.IMPEDANCE, iRecorder.Dev.TERMINATE]:
313          time.sleep(0.01)
         self.__check_dev_status()
```

2.1.21 `stop_acquisition()`

Stop data or impedance acquisition, block until data acquisition stopped or failed.

Raises:

Type	Description
`Exception`	if device not connected or acquisition stop failed.

Source code in `src\eConEXG\iRecorder\device.py`

```
283  def stop_acquisition(self) -> None:
284      """
285      Stop data or impedance acquisition, block until data acquisition stopped or failed.
286
287      Raises:
288          Exception: if device not connected or acquisition stop failed.
289      """
290      self.__check_dev_status()
291      if self.__status == iRecorder.Dev.IDLE:
292          return
293      self.__status = iRecorder.Dev.IDLE_START
294      while self.__status not in [iRecorder.Dev.IDLE, iRecorder.Dev.TERMINATE]:
295          time.sleep(0.01)
296      self.__check_dev_status()
```

2.1.22 `update_channels(channels=None)`

Update channels to acquire, invoke this method when device is not acquiring data or impedance.

Parameters:

Name	Type	Description	Default
`channels`	`Optional[dict]`	channel number and name mapping, e.g. `{0: "FPz", 1: "Oz", 2: "CPz"}`, if `None` is given, reset to all available channels with default names.	`None`

Raises:

Type	Description
`Exception`	if data/impedance acquisition in progress.

Source code in `src\eConEXG\iRecorder\device.py`

```
206     def update_channels(self, channels: Optional[dict] = None):
207         """
208         Update channels to acquire, invoke this method when device is not acquiring data or
209     impedance.
210 
211         Args:
212             channels: channel number and name mapping, e.g. `{0: "FPz", 1: "Oz", 2: "CPz"}`,
213                 if `None` is given, reset to all available channels with default names.
214 
215         Raises:
216             Exception: if data/impedance acquisition in progress.
217         """
218         if self.__status not in [iRecorder.Dev.IDLE, iRecorder.Dev.TERMINATE]:
219             warn = "Device acquisition in progress, please stop_acquisition() first."
220             raise Exception(warn)
221         if channels is None:
222             from .default_config import getChannels
223 
224             channels = getChannels(self.__dev_args["channel"])
225         self.__dev_args.update({"ch_info": channels})
226         ch_idx = [i for i in channels.keys()]
            self.__parser._update_chs(ch_idx)
```

3 iFocus

3.1 `iFocus`

Bases: `Thread`

Source code in `src\eConEXG\iFocus\data_reader.py`

```python
class iFocus(Thread):
    class Dev(Enum):
        SIGNAL = 10
        SIGNAL_START = 11
        IDLE = 30
        IDLE_START = 31
        TERMINATE = 40
        TERMINATE_START = 41

    dev_args = {
        "type": "iFocus",
        "fs_eeg": 250,
        "fs_imu": 50,
        "channel_eeg": {0: "CH0"},
        "channel_imu": {0: "X", 1: "Y", 2: "Z"},
        "AdapterInfo": "Serial Port",
    }

    def __init__(self, port: Optional[str] = None) -> None:
        """
        Args:
            port: if not given, connect to the first available device.
        """
        super().__init__(daemon=True)
        self.__status = iFocus.Dev.TERMINATE
        if port is None:
            port = iFocus.find_devs()[0]
        self.__save_data = Queue()
        self.__parser = Parser()
        self.dev_args = deepcopy(iFocus.dev_args)
        self.dev = sock(port)
        self.set_frequency()
        self.__with_q = True
        self.__socket_flag = "Device not connected, please connect first."
        self.__bdf_flag = False
        try:
            self.dev.connect_socket()
        except Exception as e:
            try:
                self.dev.close_socket()
            finally:
                raise e
        self.__status = iFocus.Dev.IDLE_START
        self.__socket_flag = None
        self._lsl_eeg = None
        self._lsl_imu = None
        self.__lsl_imu_flag = False
        self.__lsl_eeg_flag = False
        self._bdf_file = None
        self.__enable_imu = False
        self.dev_args["name"] = port
        self.start()

    def set_frequency(self, fs_eeg: int = None):
        """
        Change the sampling frequency of iFocus.

        Args:
            fs_eeg: sampling frequency of eeg data, should be 250 or 500,
                fs_imu will be automatically set to 1/5 of fs_eeg.

        Raises:
```

```
75              ValueError: if fs_eeg is not 250 or 500.
76              NotImplementedError: device firmware too old, not supporting 500Hz.
77          """
78          if self.__status == iFocus.Dev.SIGNAL:
79              raise Exception("Data acquisition already started, please stop first.")
80          if fs_eeg is None:
81              fs_eeg = self.dev_args["fs_eeg"]
82          if fs_eeg not in [250, 500]:
83              raise ValueError("fs_eeg should be 250 or 500")
84          self.dev_args["fs_eeg"] = fs_eeg
85          fs_imu = fs_eeg // 5
86          self.dev_args["fs_imu"] = fs_imu
87          if hasattr(self, "dev"):
88              self.dev.set_frequency(fs_eeg)
89
90      def get_dev_info(self) -> dict:
91          """
92          Get current device information, including device name, hardware channel number,
93  acquired channels, sample frequency, etc.
94
95          Returns:
96              A dictionary containing device information, which includes:
97                  `type`: hardware type;
98                  `channel_eeg`: channel dictionary, including EEG channel index and name;
99                  `channel_imu`: channel dictionary, including IMU channel index and name;
100                 `AdapterInfo`: adapter used for connection;
101                 `fs_eeg`: sample frequency of EEG in Hz;
102                 `fs_imu`: sample frequency of IMU in Hz;
103         """
104         return deepcopy(self.dev_args)
105
106     @staticmethod
107     def find_devs() -> list:
108         """
109         Find available iFocus devices.
110
111         Returns:
112             available device ports.
113
114         Raises:
115             Exception: if no iFocus device found.
116         """
117         return sock._find_devs()
118
119     def get_data(
120         self, timeout: Optional[float] = 0.02
121     ) -> Optional[list[Optional[list]]]:
122         """
123         Acquire all available data, make sure this function is called in a loop when `with_q`
124  is set to `True` in`start_acquisition_data()`
125
126         Args:
127             timeout: Non-negative value, blocks at most 'timeout' seconds and return, if set
128  to `None`, blocks until new data available.
129
130         Returns:
131             A list of frames, each frame is made up of 5 eeg data and 1 imu data in a shape
132  as below:
133                 [[`eeg_0`], [`eeg_1`], [`eeg_2`], [`eeg_3`], [`eeg_4`], [`imu_x`, `imu_y`,
134  `imu_z`]],
135                     in which number `0~4` after `_` indicates the time order of channel data.
136
137         Raises:
```

```
            Exception: if device not connected, connection failed, data transmission
timeout/init failed, or unknown error.

        Data Unit:
            - eeg: μV
            - imu: degree(°)
        """
        self.__check_dev_status()
        if not self.__with_q:
            return
        try:
            data: list = self.__save_data.get(timeout=timeout)
        except queue.Empty:
            return []
        while not self.__save_data.empty():
            data.extend(self.__save_data.get())
        return data

    def start_acquisition_data(self, with_q: bool = True) -> None:
        """
        Send data acquisition command to device, block until data acquisition started or
failed.

        Args:
            with_q: if True, signal data will be stored in a queue and **should** be acquired
by calling `get_data()` in a loop in case data queue is full.
                if False, new data will not be directly available and can only be acquired
through lsl stream.

        """
        self.__check_dev_status()
        self.__with_q = with_q
        if self.__status == iFocus.Dev.SIGNAL:
            return
        self.__status = iFocus.Dev.SIGNAL_START
        while self.__status not in [iFocus.Dev.SIGNAL, iFocus.Dev.TERMINATE]:
            time.sleep(0.01)
        self.__check_dev_status()

    def stop_acquisition(self) -> None:
        """
        Stop data or impedance acquisition, block until data acquisition stopped or failed.
        """
        self.__check_dev_status()
        self.__status = iFocus.Dev.IDLE_START
        while self.__status not in [iFocus.Dev.IDLE, iFocus.Dev.TERMINATE]:
            time.sleep(0.01)
        self.__check_dev_status()

    def open_lsl_eeg(self):
        """
        Open LSL EEG stream, can be invoked after `start_acquisition_data()`.

        Raises:
            Exception: if data acquisition not started or LSL stream already opened.
            LSLException: if LSL stream creation failed.
            importError: if `pylsl` is not installed or liblsl not installed for unix like
system.
        """
        if self.__status != iFocus.Dev.SIGNAL:
            raise Exception("Data acquisition not started, please start first.")
        if hasattr(self, "_lsl_eeg"):
            raise Exception("LSL stream already opened.")
```

```
        from ..utils.lslWrapper import lslSender

        self._lsl_eeg = lslSender(
            self.dev_args["channel_eeg"],
            f"{self.dev_args['type']}EEG{self.dev_args['name'][-2:]}",
            "EEG",
            self.dev_args["fs_eeg"],
            with_trigger=False,
        )
        self.__lsl_eeg_flag = True

    def close_lsl_eeg(self):
        """
        Close LSL EEG stream manually, invoked automatically after `stop_acquisition()` and
`close_dev()`
        """
        self.__lsl_eeg_flag = False
        if hasattr(self, "_lsl_eeg"):
            del self._lsl_eeg

    def open_lsl_imu(self):
        """
        Open LSL IMU stream, can be invoked after `start_acquisition_data()`.

        Raises:
            Exception: if data acquisition not started or LSL stream already opened.
            LSLException: if LSL stream creation failed.
            importError: if `pylsl` is not installed or liblsl not installed for unix like
system.
        """
        if self.__status != iFocus.Dev.SIGNAL:
            raise Exception("Data acquisition not started, please start first.")
        if hasattr(self, "_lsl_imu"):
            raise Exception("LSL stream already opened.")
        from ..utils.lslWrapper import lslSender

        self._lsl_imu = lslSender(
            self.dev_args["channel_imu"],
            f"{self.dev_args['type']}IMU{self.dev_args['name'][-2:]}",
            "IMU",
            self.dev_args["fs_imu"],
            unit="degree",
            with_trigger=False,
        )
        self.__lsl_imu_flag = True

    def close_lsl_imu(self):
        """
        Close LSL IMU stream manually, invoked automatically after `stop_acquisition()` and
`close_dev()`
        """
        self.__lsl_imu_flag = False
        if hasattr(self, "_lsl_imu"):
            del self._lsl_imu

    def setIMUFlag(self, check):
        self.__enable_imu = check

    def create_bdf_file(self, filename: str):
        """
        Create a BDF file and save data to it, invoke it after `start_acquisition_data()`.

        Args:
```

```
264             filename: file name to save data, accept absolute or relative path.
265 
266         Raises:
267             Exception: if data acquisition not started or `save_bdf_file` is invoked and BDF
268 file already created.
269             OSError: if BDF file creation failed, this may be caused by invalid file path or
270 permission issue.
271             importError: if `pyedflib` is not installed.
272         """
273         if self.__status != iFocus.Dev.SIGNAL:
274             raise Exception("Data acquisition not started")
275         if self._bdf_file is not None:
276             raise Exception("BDF file already created.")
277         from ..utils.bdfWrapper import bdfSaverEEG, bdfSaverEEGIMU
278 
279         if filename[-4:].lower() != ".bdf":
280             filename += ".bdf"
281         if self.__enable_imu:
282             self._bdf_file = bdfSaverEEGIMU(
283                 filename,
284                 self.dev_args["channel_eeg"],
285                 self.dev_args["fs_eeg"],
286                 self.dev_args["channel_imu"],
287                 self.dev_args["fs_imu"],
288                 self.dev_args["type"],
289             )
290         else:
291             self._bdf_file = bdfSaverEEG(
292                 filename,
293                 self.dev_args["channel_eeg"],
294                 self.dev_args["fs_eeg"],
295                 self.dev_args["type"],
296             )
297         self.__bdf_flag = True
298 
299     def close_bdf_file(self):
300         """
301         Close and save BDF file manually, invoked automatically after `stop_acquisition()` or
302 `close_dev()`
303         """
304         self.__bdf_flag = False
305         if self._bdf_file is not None:
306             self._bdf_file.close_bdf()
307             del self._bdf_file
308 
309     def send_bdf_marker(self, marker: str):
310         """
311         Send marker to BDF file, can be invoked after `create_bdf_file()`, otherwise it will
312 be ignored.
313 
314         Args:
315             marker: marker string to write.
316         """
317         if hasattr(self, "_bdf_file"):
318             self._bdf_file.write_Annotation(marker)
319 
320     def close_dev(self):
321         """
322         Close device connection and release resources.
323         """
324         if self.__status != iFocus.Dev.TERMINATE:
325             # ensure socket is closed correctly
326             self.__status = iFocus.Dev.TERMINATE_START
```

```
            while self.__status != iFocus.Dev.TERMINATE:
                time.sleep(0.1)
        if self.is_alive():
            self.join()

    def __recv_data(self):
        try:
            self.dev.start_data()
            self.__status = iFocus.Dev.SIGNAL
        except Exception:
            self.__socket_flag = "SIGNAL mode initialization failed."
            self.__status = iFocus.Dev.TERMINATE_START

        while self.__status in [iFocus.Dev.SIGNAL]:
            try:
                data = self.dev.recv_socket()
                if not data:
                    raise Exception("Data transmission timeout.")
                ret = self.__parser.parse_data(data)
                if ret:
                    if self.__lsl_eeg_flag:
                        self._lsl_eeg.push_chunk(
                            [frame for frames in ret for frame in frames[:-1]]
                        )
                    if self.__lsl_imu_flag:
                        self._lsl_imu.push_chunk([frame[-1] for frame in ret])
                    if self.__with_q:
                        self.__save_data.put(ret)
                    if self.__bdf_flag:
                        self._bdf_file.write_chunk(ret)
            except Exception as e:
                print(e)
                self.__socket_flag = "Data transmission timeout."
                self.__status = iFocus.Dev.TERMINATE_START

        # clear buffer
        self.close_lsl_eeg()
        self.close_lsl_imu()
        self.close_bdf_file()
        # self.dev.stop_recv()
        self.__parser.clear_buffer()
        self.__save_data.put(None)
        while self.__save_data.get() is not None:
            continue
        # stop recv data
        if self.__status != iFocus.Dev.TERMINATE_START:
            try:  # stop data acquisition when thread ended
                self.dev.stop_recv()
            except Exception:
                if self.__status == iFocus.Dev.IDLE_START:
                    self.__socket_flag = "Connection lost."
                self.__status = iFocus.Dev.TERMINATE_START

    def run(self):
        while self.__status != iFocus.Dev.TERMINATE_START:
            if self.__status == iFocus.Dev.SIGNAL_START:
                self.__recv_data()
            elif self.__status == iFocus.Dev.IDLE_START:
                self.__status = iFocus.Dev.IDLE
                while self.__status == iFocus.Dev.IDLE:
                    time.sleep(0.1)
            else:
                self.__socket_flag = f"Unknown status: {self.__status.name}"
```

```
                    break
            try:
                self.dev.close_socket()
            finally:
                self.__status = iFocus.Dev.TERMINATE

        def __check_dev_status(self):
            if self.__socket_flag is None:
                return
            if self.is_alive():
                self.close_dev()
            raise Exception(str(self.__socket_flag))
```

3.1.1 `__init__(port=None)`

Parameters:

Name	Type	Description	Default
`port`	`Optional[str]`	if not given, connect to the first available device.	`None`

Source code in `src\eConEXG\iFocus\data_reader.py`

```
31  def __init__(self, port: Optional[str] = None) -> None:
32      """
33      Args:
34          port: if not given, connect to the first available device.
35      """
36      super().__init__(daemon=True)
37      self.__status = iFocus.Dev.TERMINATE
38      if port is None:
39          port = iFocus.find_devs()[0]
40      self.__save_data = Queue()
41      self.__parser = Parser()
42      self.dev_args = deepcopy(iFocus.dev_args)
43      self.dev = sock(port)
44      self.set_frequency()
45      self.__with_q = True
46      self.__socket_flag = "Device not connected, please connect first."
47      self.__bdf_flag = False
48      try:
49          self.dev.connect_socket()
50      except Exception as e:
51          try:
52              self.dev.close_socket()
53          finally:
54              raise e
55      self.__status = iFocus.Dev.IDLE_START
56      self.__socket_flag = None
57      self._lsl_eeg = None
58      self._lsl_imu = None
59      self.__lsl_imu_flag = False
60      self.__lsl_eeg_flag = False
61      self._bdf_file = None
62      self.__enable_imu = False
63      self.dev_args["name"] = port
64      self.start()
```

3.1.2 `close_bdf_file()`

Close and save BDF file manually, invoked automatically after `stop_acquisition()` or `close_dev()`

Source code in `src\eConEXG\iFocus\data_reader.py`

```
284 def close_bdf_file(self):
285     """
286     Close and save BDF file manually, invoked automatically after `stop_acquisition()` or
287 `close_dev()`
288     """
289     self.__bdf_flag = False
290     if self._bdf_file is not None:
291         self._bdf_file.close_bdf()
            del self._bdf_file
```

3.1.3 `close_dev()`

Close device connection and release resources.

Source code in `src\eConEXG\iFocus\data_reader.py`

```
303   def close_dev(self):
304       """
305       Close device connection and release resources.
306       """
307       if self.__status != iFocus.Dev.TERMINATE:
308           # ensure socket is closed correctly
309           self.__status = iFocus.Dev.TERMINATE_START
310           while self.__status != iFocus.Dev.TERMINATE:
311               time.sleep(0.1)
312       if self.is_alive():
313           self.join()
```

3.1.4 `close_lsl_eeg()`

Close LSL EEG stream manually, invoked automatically after `stop_acquisition()` and `close_dev()`

Source code in `src\eConEXG\iFocus\data_reader.py`

```
202   def close_lsl_eeg(self):
203       """
204       Close LSL EEG stream manually, invoked automatically after `stop_acquisition()` and
205   `close_dev()`
206       """
207       self.__lsl_eeg_flag = False
208       if hasattr(self, "_lsl_eeg"):
              del self._lsl_eeg
```

3.1.5 `close_lsl_imu()`

Close LSL IMU stream manually, invoked automatically after `stop_acquisition()` and `close_dev()`

Source code in `src\eConEXG\iFocus\data_reader.py`

```
235   def close_lsl_imu(self):
236       """
237       Close LSL IMU stream manually, invoked automatically after `stop_acquisition()` and
238   `close_dev()`
239       """
240       self.__lsl_imu_flag = False
241       if hasattr(self, "_lsl_imu"):
              del self._lsl_imu
```

3.1.6 `create_bdf_file(filename)`

Create a BDF file and save data to it, invoke it after `start_acquisition_data()`.

Parameters:

Name	Type	Description	Default
`filename`	`str`	file name to save data, accept absolute or relative path.	*required*

Raises:

Type	Description
`Exception`	if data acquisition not started or `save_bdf_file` is invoked and BDF file already created.
`OSError`	if BDF file creation failed, this may be caused by invalid file path or permission issue.
`importError`	if `pyedflib` is not installed.

Source code in `src\eConEXG\iFocus\data_reader.py`

```
246     def create_bdf_file(self, filename: str):
247         """
248         Create a BDF file and save data to it, invoke it after `start_acquisition_data()`.
249 
250         Args:
251             filename: file name to save data, accept absolute or relative path.
252 
253         Raises:
254             Exception: if data acquisition not started or `save_bdf_file` is invoked and BDF file
255     already created.
256             OSError: if BDF file creation failed, this may be caused by invalid file path or
257     permission issue.
258             importError: if `pyedflib` is not installed.
259         """
260         if self.__status != iFocus.Dev.SIGNAL:
261             raise Exception("Data acquisition not started")
262         if self._bdf_file is not None:
263             raise Exception("BDF file already created.")
264         from ..utils.bdfWrapper import bdfSaverEEG, bdfSaverEEGIMU
265 
266         if filename[-4:].lower() != ".bdf":
267             filename += ".bdf"
268         if self.__enable_imu:
269             self._bdf_file = bdfSaverEEGIMU(
270                 filename,
271                 self.dev_args["channel_eeg"],
272                 self.dev_args["fs_eeg"],
273                 self.dev_args["channel_imu"],
274                 self.dev_args["fs_imu"],
275                 self.dev_args["type"],
276             )
277         else:
278             self._bdf_file = bdfSaverEEG(
279                 filename,
280                 self.dev_args["channel_eeg"],
281                 self.dev_args["fs_eeg"],
282                 self.dev_args["type"],
            )
        self.__bdf_flag = True
```

3.1.7 `find_devs()` `staticmethod`

Find available iFocus devices.

Returns:

Type	Description
`list`	available device ports.

Raises:

Type	Description
`Exception`	if no iFocus device found.

Source code in `src\eConEXG\iFocus\data_reader.py`

```
105     @staticmethod
106     def find_devs() -> list:
107         """
108         Find available iFocus devices.
109 
110         Returns:
111             available device ports.
112 
113         Raises:
114             Exception: if no iFocus device found.
115         """
116         return sock._find_devs()
```

3.1.8 `get_data(timeout=0.02)`

Acquire all available data, make sure this function is called in a loop when `with_q` is set to `True` in `start_acquisition_data()`

Parameters:

Name	Type	Description	Default
`timeout`	`Optional[float]`	Non-negative value, blocks at most 'timeout' seconds and return, if set to `None`, blocks until new data available.	`0.02`

Returns:

Type	Description
`Optional[list[Optional[list]]]`	A list of frames, each frame is made up of 5 eeg data and 1 imu data in a shape as below: [[`eeg_0`], [`eeg_1`], [`eeg_2`], [`eeg_3`], [`eeg_4`], [`imu_x` , `imu_y` , `imu_z`]], in which number `0~4` after `_` indicates the time order of channel data.

Raises:

Type	Description
`Exception`	if device not connected, connection failed, data transmission timeout/init failed, or unknown error.

Data Unit

- eeg: μV
- imu: degree(°)

Source code in `src\eConEXG\iFocus\data_reader.py`

```python
def get_data(
    self, timeout: Optional[float] = 0.02
) -> Optional[list[Optional[list]]]:
    """
    Acquire all available data, make sure this function is called in a loop when `with_q` is
set to `True` in`start_acquisition_data()`

    Args:
        timeout: Non-negative value, blocks at most 'timeout' seconds and return, if set to
`None`, blocks until new data available.

    Returns:
        A list of frames, each frame is made up of 5 eeg data and 1 imu data in a shape as
below:
            [[`eeg_0`], [`eeg_1`], [`eeg_2`], [`eeg_3`], [`eeg_4`], [`imu_x`, `imu_y`,
`imu_z`]],
                in which number `0~4` after `_` indicates the time order of channel data.

    Raises:
        Exception: if device not connected, connection failed, data transmission timeout/init
failed, or unknown error.

    Data Unit:
        - eeg: μV
        - imu: degree(°)
    """
    self.__check_dev_status()
    if not self.__with_q:
        return
    try:
        data: list = self.__save_data.get(timeout=timeout)
    except queue.Empty:
        return []
    while not self.__save_data.empty():
        data.extend(self.__save_data.get())
    return data
```

3.1.9 `get_dev_info()`

Get current device information, including device name, hardware channel number, acquired channels, sample frequency, etc.

Returns:

Type	Description
`dict`	A dictionary containing device information, which includes: `type`: hardware type; `channel_eeg`: channel dictionary, including EEG channel index and name; `channel_imu`: channel dictionary, including IMU channel index and name; `AdapterInfo`: adapter used for connection; `fs_eeg`: sample frequency of EEG in Hz; `fs_imu`: sample frequency of IMU in Hz;

Source code in `src\eConEXG\iFocus\data_reader.py`

```
 90  def get_dev_info(self) -> dict:
 91      """
 92      Get current device information, including device name, hardware channel number, acquired
 93  channels, sample frequency, etc.
 94
 95      Returns:
 96          A dictionary containing device information, which includes:
 97              `type`: hardware type;
 98              `channel_eeg`: channel dictionary, including EEG channel index and name;
 99              `channel_imu`: channel dictionary, including IMU channel index and name;
100              `AdapterInfo`: adapter used for connection;
101              `fs_eeg`: sample frequency of EEG in Hz;
102              `fs_imu`: sample frequency of IMU in Hz;
103      """
          return deepcopy(self.dev_args)
```

3.1.10 `open_lsl_eeg()`

Open LSL EEG stream, can be invoked after `start_acquisition_data()`.

Raises:

Type	Description
`Exception`	if data acquisition not started or LSL stream already opened.
`LSLException`	if LSL stream creation failed.
`importError`	if `pylsl` is not installed or liblsl not installed for unix like system.

Source code in `src\eConEXG\iFocus\data_reader.py`

```
178   def open_lsl_eeg(self):
179       """
180       Open LSL EEG stream, can be invoked after `start_acquisition_data()`.
181
182       Raises:
183           Exception: if data acquisition not started or LSL stream already opened.
184           LSLException: if LSL stream creation failed.
185           importError: if `pylsl` is not installed or liblsl not installed for unix like
186   system.
187       """
188       if self.__status != iFocus.Dev.SIGNAL:
189           raise Exception("Data acquisition not started, please start first.")
190       if hasattr(self, "_lsl_eeg"):
191           raise Exception("LSL stream already opened.")
192       from ..utils.lslWrapper import lslSender
193
194       self._lsl_eeg = lslSender(
195           self.dev_args["channel_eeg"],
196           f"{self.dev_args['type']}EEG{self.dev_args['name'][-2:]}",
197           "EEG",
198           self.dev_args["fs_eeg"],
199           with_trigger=False,
200       )
          self.__lsl_eeg_flag = True
```

3.1.11 `open_lsl_imu()`

Open LSL IMU stream, can be invoked after `start_acquisition_data()`.

Raises:

Type	Description
`Exception`	if data acquisition not started or LSL stream already opened.
`LSLException`	if LSL stream creation failed.
`importError`	if `pylsl` is not installed or liblsl not installed for unix like system.

Source code in `src\eConEXG\iFocus\data_reader.py`

```
210  def open_lsl_imu(self):
211      """
212      Open LSL IMU stream, can be invoked after `start_acquisition_data()`.
213
214      Raises:
215          Exception: if data acquisition not started or LSL stream already opened.
216          LSLException: if LSL stream creation failed.
217          importError: if `pylsl` is not installed or liblsl not installed for unix like
218  system.
219      """
220      if self.__status != iFocus.Dev.SIGNAL:
221          raise Exception("Data acquisition not started, please start first.")
222      if hasattr(self, "_lsl_imu"):
223          raise Exception("LSL stream already opened.")
224      from ..utils.lslWrapper import lslSender
225
226      self._lsl_imu = lslSender(
227          self.dev_args["channel_imu"],
228          f"{self.dev_args['type']}IMU{self.dev_args['name'][-2:]}",
229          "IMU",
230          self.dev_args["fs_imu"],
231          unit="degree",
232          with_trigger=False,
233      )
          self.__lsl_imu_flag = True
```

3.1.12 `send_bdf_marker(marker)`

Send marker to BDF file, can be invoked after `create_bdf_file()`, otherwise it will be ignored.

Parameters:

Name	Type	Description	Default
`marker`	`str`	marker string to write.	*required*

Source code in `src\eConEXG\iFocus\data_reader.py`

```
293  def send_bdf_marker(self, marker: str):
294      """
295      Send marker to BDF file, can be invoked after `create_bdf_file()`, otherwise it will be
296  ignored.
297
298      Args:
299          marker: marker string to write.
300      """
301      if hasattr(self, "_bdf_file"):
             self._bdf_file.write_Annotation(marker)
```

3.1.13 `set_frequency(fs_eeg=None)`

Change the sampling frequency of iFocus.

Parameters:

Name	Type	Description	Default
`fs_eeg`	`int`	sampling frequency of eeg data, should be 250 or 500, fs_imu will be automatically set to 1/5 of fs_eeg.	`None`

Raises:

Type	Description
`ValueError`	if fs_eeg is not 250 or 500.
`NotImplementedError`	device firmware too old, not supporting 500Hz.

Source code in `src\eConEXG\iFocus\data_reader.py`

```
66  def set_frequency(self, fs_eeg: int = None):
67      """
68      Change the sampling frequency of iFocus.
69
70      Args:
71          fs_eeg: sampling frequency of eeg data, should be 250 or 500,
72              fs_imu will be automatically set to 1/5 of fs_eeg.
73
74      Raises:
75          ValueError: if fs_eeg is not 250 or 500.
76          NotImplementedError: device firmware too old, not supporting 500Hz.
77      """
78      if self.__status == iFocus.Dev.SIGNAL:
79          raise Exception("Data acquisition already started, please stop first.")
80      if fs_eeg is None:
81          fs_eeg = self.dev_args["fs_eeg"]
82      if fs_eeg not in [250, 500]:
83          raise ValueError("fs_eeg should be 250 or 500")
84      self.dev_args["fs_eeg"] = fs_eeg
85      fs_imu = fs_eeg // 5
86      self.dev_args["fs_imu"] = fs_imu
87      if hasattr(self, "dev"):
88          self.dev.set_frequency(fs_eeg)
```

3.1.14 `start_acquisition_data(with_q=True)`

Send data acquisition command to device, block until data acquisition started or failed.

Parameters:

Name	Type	Description	Default
`with_q`	`bool`	if True, signal data will be stored in a queue and **should** be acquired by calling `get_data()` in a loop in case data queue is full. if False, new data will not be directly available and can only be acquired through lsl stream.	`True`

Source code in `src\eConEXG\iFocus\data_reader.py`

```
150  def start_acquisition_data(self, with_q: bool = True) -> None:
151      """
152      Send data acquisition command to device, block until data acquisition started or failed.
153
154      Args:
155          with_q: if True, signal data will be stored in a queue and **should** be acquired by
156  calling `get_data()` in a loop in case data queue is full.
157              if False, new data will not be directly available and can only be acquired
158  through lsl stream.
159
160      """
161      self.__check_dev_status()
162      self.__with_q = with_q
163      if self.__status == iFocus.Dev.SIGNAL:
164          return
165      self.__status = iFocus.Dev.SIGNAL_START
166      while self.__status not in [iFocus.Dev.SIGNAL, iFocus.Dev.TERMINATE]:
             time.sleep(0.01)
         self.__check_dev_status()
```

3.1.15 `stop_acquisition()`

Stop data or impedance acquisition, block until data acquisition stopped or failed.

Source code in `src\eConEXG\iFocus\data_reader.py`

```
168  def stop_acquisition(self) -> None:
169      """
170      Stop data or impedance acquisition, block until data acquisition stopped or failed.
171      """
172      self.__check_dev_status()
173      self.__status = iFocus.Dev.IDLE_START
174      while self.__status not in [iFocus.Dev.IDLE, iFocus.Dev.TERMINATE]:
175          time.sleep(0.01)
176      self.__check_dev_status()
```

I.I Trigger Box

4 Wireless

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```
class triggerBoxWireless:
    def __init__(self, port: str = None):
        """
        Args:
            port: The serial port of the trigger box. If not given,
                the function will try to find the trigger box automatically.

        Raises:
            Exception: If the trigger box is not found.
        """
        from serial import Serial
        from serial.tools.list_ports import comports

        if not port:
            for ports in comports():
                if ports.pid == 0x6001 and ports.vid == 0x0403:
                    port = ports.device
                    break
            else:
                raise Exception("Trigger box not found")
        self.dev = Serial(port, baudrate=115200, timeout=1)
        self.__last_timestamp = time.perf_counter()
        self.__warn = "Marker interval too short, amplifier may fail to receive it. Suggested
interval is above 50ms"
        time.sleep(0.1)

    def sendMarker(self, marker: int):
        """
        Send a marker to the trigger box.

        Args:
            marker: range from `1` to `255`, `13` is not available and reserved for internal
use.

        Raises:
            Exception: If the marker is invalid.
        """
        if time.perf_counter() - self.__last_timestamp < 0.04:
            print(self.__warn)
        if not isinstance(marker, int):
            marker = int(marker)
        if marker == 13 or marker <= 0 or marker > 255:
            raise Exception("Invalid marker")
        marker = marker.to_bytes(length=1, byteorder="big", signed=False)
        self.dev.write(marker + b"\x55\x66\x0d")
        self.__last_timestamp = time.perf_counter()

    def close_dev(self):
        self.dev.close()
```

4.1 `__init__(port=None)`

Parameters:

Name	Type	Description	Default
`port`	`str`	The serial port of the trigger box. If not given, the function will try to find the trigger box automatically.	`None`

Raises:

Type	Description
`Exception`	If the trigger box is not found.

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```
6   def __init__(self, port: str = None):
7       """
8       Args:
9           port: The serial port of the trigger box. If not given,
10              the function will try to find the trigger box automatically.
11
12      Raises:
13          Exception: If the trigger box is not found.
14      """
15      from serial import Serial
16      from serial.tools.list_ports import comports
17
18      if not port:
19          for ports in comports():
20              if ports.pid == 0x6001 and ports.vid == 0x0403:
21                  port = ports.device
22                  break
23          else:
24              raise Exception("Trigger box not found")
25      self.dev = Serial(port, baudrate=115200, timeout=1)
26      self.__last_timestamp = time.perf_counter()
27      self.__warn = "Marker interval too short, amplifier may fail to receive it. Suggested
28  interval is above 50ms"
        time.sleep(0.1)
```

4.2 `sendMarker(marker)`

Send a marker to the trigger box.

Parameters:

Name	Type	Description	Default
`marker`	`int`	range from `1` to `255`, `13` is not available and reserved for internal use.	*required*

Raises:

Type	Description
`Exception`	If the marker is invalid.

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```
30  def sendMarker(self, marker: int):
31      """
32      Send a marker to the trigger box.
33
34      Args:
35          marker: range from `1` to `255`, `13` is not available and reserved for internal use.
36
37      Raises:
38          Exception: If the marker is invalid.
39      """
40      if time.perf_counter() - self.__last_timestamp < 0.04:
41          print(self.__warn)
42      if not isinstance(marker, int):
43          marker = int(marker)
44      if marker == 13 or marker <= 0 or marker > 255:
45          raise Exception("Invalid marker")
46      marker = marker.to_bytes(length=1, byteorder="big", signed=False)
47      self.dev.write(marker + b"\x55\x66\x0d")
48      self.__last_timestamp = time.perf_counter()
```

5 Wired

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```python
class triggerBoxWired:
    def __init__(self, port: str = None):
        """
        Args:
            port: The serial port of the trigger box. If not given,
                the function will try to find the trigger box automatically.

        Raises:
            Exception: If the trigger box is not found.
        """
        from serial import Serial
        from serial.tools.list_ports import comports

        if not port:
            for ports in comports():
                if ports.pid == 0x5740 and ports.vid == 0x0483:
                    port = ports.device
                    break
            else:
                raise Exception("Trigger box not found")
        self.dev = Serial(port, timeout=1)

    def sendMarker(self, marker: int):
        """
        Send a marker to the trigger box.

        Args:
            marker: range from `1` to `255`.

        Raises:
            Exception: If the marker is invalid.
        """
        if not isinstance(marker, int):
            marker = int(marker)
        if marker <= 0 or marker > 255:
            raise Exception("Invalid marker")
        self.dev.write(marker.to_bytes(length=1, byteorder="big", signed=False))

    def close_dev(self):
        self.dev.close()
```

5.1 `__init__(port=None)`

Parameters:

Name	Type	Description	Default
`port`	`str`	The serial port of the trigger box. If not given, the function will try to find the trigger box automatically.	`None`

Raises:

Type	Description
`Exception`	If the trigger box is not found.

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```
55  def __init__(self, port: str = None):
56      """
57      Args:
58          port: The serial port of the trigger box. If not given,
59              the function will try to find the trigger box automatically.
60
61      Raises:
62          Exception: If the trigger box is not found.
63      """
64      from serial import Serial
65      from serial.tools.list_ports import comports
66
67      if not port:
68          for ports in comports():
69              if ports.pid == 0x5740 and ports.vid == 0x0483:
70                  port = ports.device
71                  break
72          else:
73              raise Exception("Trigger box not found")
74      self.dev = Serial(port, timeout=1)
```

5.2 `sendMarker(marker)`

Send a marker to the trigger box.

Parameters:

Name	Type	Description	Default
`marker`	`int`	range from `1` to `255`.	*required*

Raises:

Type	Description
`Exception`	If the marker is invalid.

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```python
def sendMarker(self, marker: int):
    """
    Send a marker to the trigger box.

    Args:
        marker: range from `1` to `255`.

    Raises:
        Exception: If the marker is invalid.
    """
    if not isinstance(marker, int):
        marker = int(marker)
    if marker <= 0 or marker > 255:
        raise Exception("Invalid marker")
    self.dev.write(marker.to_bytes(length=1, byteorder="big", signed=False))
```

6 Light Stimulator

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```python
class lightStimulator:
    def __init__(self, port: str = None):
        from serial import Serial
        from serial.tools.list_ports import comports

        self.wait_time = 0.1
        self.channels = 6

        if not port:
            for ports in comports():
                if (
                    ports.pid == 0x6001
                    and ports.vid == 0x0403
                    and ports.serial_number in ["LIGHTSTIMA", "LIGHTSTIM"]
                ):
                    port = ports.device
                    break
            else:
                raise Exception("Light stimulator not found")
        self.dev = Serial(port, baudrate=115200, timeout=2)
        self.dev.read_all()

    def vep_mode(self, fs: list[Optional[float]] = [1, 1, 1, 1, 1, 1]):
        """
        Enter VEP mode, which allows you to control the frequency of each channel separately.

        Args:
            fs: List of frequencies in Hz, range from 0 to 100 with 0.1Hz resolution.
                If a corresponding frequency is None, 0 or not given, it will be set to off.

        Raises:
            Exception: If the frequency is invalid or hardware error.
        """
        fss = fs.copy()
        if len(fss) < self.channels:
            fss += [0] * (self.channels - len(fss))
        for i in range(self.channels):
            fss[i] = self._validate_fs(fss[i])
        command = ",".join([f"{f:.1f}" for f in fss[: self.channels]])
        command = f"AT+VEP={command}\r\n".encode()
        self.dev.write(command)
        time.sleep(self.wait_time)
        ret = self.dev.read_all()
        if b"SSVEP MODE OK" not in ret:
            raise Exception("Failed to set VEP mode")

    def erp_mode(self, fs: float):
        """
        Enter ERP mode, which allows you to control the frequency of all channels at once.

        Args:
            fs: Frequency in Hz, range from 0 to 100 with 0.1Hz resolution.

        Raises:
            Exception: If the frequency is invalid or hardware error.
        """
        fs = self._validate_fs(fs)
        command = f"AT+ERP={fs:.1f}\r\n".encode()
        self.dev.write(command)
        time.sleep(self.wait_time)
        ret = self.dev.read_all()
        if b"ERP MODE OK" not in ret:
```

```
158             raise Exception("Failed to set VEP mode")
159
160     def _validate_fs(self, fs: Optional[float]):
161         if not isinstance(fs, (int, float)):
162             if fs is None:
163                 fs = 0
164             else:
165                 raise Exception("Invalid frequency")
166         if fs > 100 or fs < 0:
167             raise Exception("Invalid frequency")
168         return fs
169
170     def close_dev(self):
171         self.dev.close()
```

6.1 `erp_mode(fs)`

Enter ERP mode, which allows you to control the frequency of all channels at once.

Parameters:

Name	Type	Description	Default
`fs`	`float`	Frequency in Hz, range from 0 to 100 with 0.1Hz resolution.	*required*

Raises:

Type	Description
`Exception`	If the frequency is invalid or hardware error.

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```
142  def erp_mode(self, fs: float):
143      """
144      Enter ERP mode, which allows you to control the frequency of all channels at once.
145
146      Args:
147          fs: Frequency in Hz, range from 0 to 100 with 0.1Hz resolution.
148
149      Raises:
150          Exception: If the frequency is invalid or hardware error.
151      """
152      fs = self._validate_fs(fs)
153      command = f"AT+ERP={fs:.1f}\r\n".encode()
154      self.dev.write(command)
155      time.sleep(self.wait_time)
156      ret = self.dev.read_all()
157      if b"ERP MODE OK" not in ret:
158          raise Exception("Failed to set VEP mode")
```

6.2 `vep_mode(fs=[1, 1, 1, 1, 1, 1])`

Enter VEP mode, which allows you to control the frequency of each channel separately.

Parameters:

Name	Type	Description	Default
`fs`	`list[Optional[float]]`	List of frequencies in Hz, range from 0 to 100 with 0.1Hz resolution. If a corresponding frequency is None, 0 or not given, it will be set to off.	`[1, 1, 1, 1, 1, 1]`

Raises:

Type	Description
`Exception`	If the frequency is invalid or hardware error.

Source code in `src\eConEXG\triggerBox\triggerbox.py`

```python
def vep_mode(self, fs: list[Optional[float]] = [1, 1, 1, 1, 1, 1]):
    """
    Enter VEP mode, which allows you to control the frequency of each channel separately.

    Args:
        fs: List of frequencies in Hz, range from 0 to 100 with 0.1Hz resolution.
            If a corresponding frequency is None, 0  or not given, it will be set to off.

    Raises:
        Exception: If the frequency is invalid or hardware error.
    """
    fss = fs.copy()
    if len(fss) < self.channels:
        fss += [0] * (self.channels - len(fss))
    for i in range(self.channels):
        fss[i] = self._validate_fs(fss[i])
    command = ",".join([f"{f:.1f}" for f in fss[: self.channels]])
    command = f"AT+VEP={command}\r\n".encode()
    self.dev.write(command)
    time.sleep(self.wait_time)
    ret = self.dev.read_all()
    if b"SSVEP MODE OK" not in ret:
        raise Exception("Failed to set VEP mode")
```

7 Changelog

Here you can find all the released changes to eConEXG.

7.1 0.1.18

Release on 2024-10-10

- **ADD** Add `create_bdf_file` and `close_bdf_file` to iRecorder and eConAlpha.
- **ADD** Add imu support to iRecorder and eConAlpha.

7.2 0.1.15

- **ADD** eConAlpha device SDK

7.3 0.1.14

Released on 2024-08-08.

- **Update** pyEDFlib dependency requirement in `pyproject.toml` from `0.1.37` to `0.1.38` to support `numpy>=2.0.0`.
- **Optimize** `set_frequency()` in `iRecorder`, now you can set sample rate after device connection.

- **Fix** `iFocus` not raise Exception after lost connection with USBadapter.
- **Change** `with_q` argument in `iRecorder` and `iFocus` from constructor to `start_acquisition_data()`.
- **Deprecate** `save_bdf_file()` in `iRecorder`, use `create_bdf_file()` instead.

7.4 0.1.13

Released on 2024-08-01.

- **Add** `get_dev_info()` in `iFocus` class to get device information.
- **Add** selectable `500Hz` eeg and corresponding `100Hz` IMU sampling rate in `iFocus` class through `set_frequency()`.
- **Add** `with_q = False` option in `iRecorder` and `iFocus` constructor to drop the necessity of loop calling `get_data()` in SIGNAL mode.
- **Add** `__version__` field of eConEXG package to check the package version, it can be accessed throught `eConEXG.__version__`.

- **Fix** last valid packet number in `iFocus` warning message wrongly displayed as a fixed number issue.
- **Change** the default data parse length from 0.01 seconds to 10 frames in `iRecorder` to match hardware settings.
- **Improve** the aesthetics of a document interface.

7.5 0.1.12

Released on 2024-07-25.

- **Fix** equipment format warning issue on bdf save.
- **Fix** Rounding physical_max value of bdf file， resulting to more accurate data precision.
- **Fix** `sendMarker()` function not working in `triggerBox` class when python version<3.11.

7.6 0.1.11

Released on 2024-07-12.

- **Update** documentation homepage.

7.7 0.1.10

Released on 2024-07-08.

- **Add** lsl support for iFocus.
- **Add** support for iRecorder 16-channel wired mode.

- **Change** default timeout of `get_data()` function in `iRecorder` from `None` to `0.02`.

7.8 0.1.09

Released on 2024-06-28.

- **Add** support for iRecorder 8-channel wired mode.