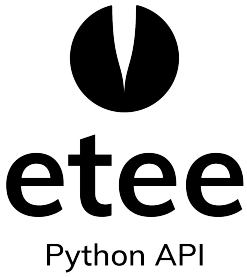
etee Python API

Pilar Zhang Qiu, Dimitri Chikhladze

Apr 26, 2023

CONTENTS



etee is the first-of-its-kind finger, gesture and pressure sensing controller. The etee Python API library allows developers to work with eteeControllers. Currently, Python programming language is supported.

 For the Python API code and installation, visit our Github repository here.

This documentation describes how to wirelessly interface with the eteeControllers via an eteeDongle.

Some features include:

- Connecting or disconnecting devices
- Starting or stopping the streaming of sensor data
- Retrieving tactile, IMU and device state data

CHAPTER

ONE

LIST OF CONTENT

1.1 1. Overview

On this page

1.1.1 1.1. Hardware

etee hardware consists of one eteeDongle and two eteeControllers (left and right hand). Optionally, etee trackers can be attached to eteeControllers for VR use.

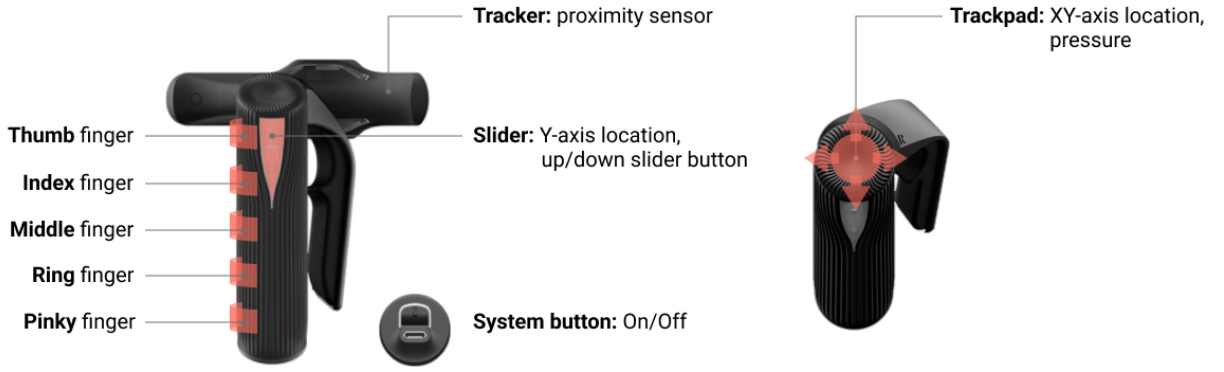
An eteeDongle connects to a computer through a serial port, while the eteeControllers connect to an eteeDongle through BLE communication.

The eteeControllers use **TG0's sensing technology** to provide tactile data. This tactile data tracks the movement and pressure from all five fingers, and provides additional inputs such as trackpad XY positional data. In-built IMU sensors also allow for the detection of the eteeController translation and rotation. These tactile and IMU data are included into a data packet, which is sent by the eteeControllers to the eteeDongle. This packet can then be accessed through serial communication with the eteeDongle. Data packets from each eteeController are streamed at a rate of approximately 100 packets/second.

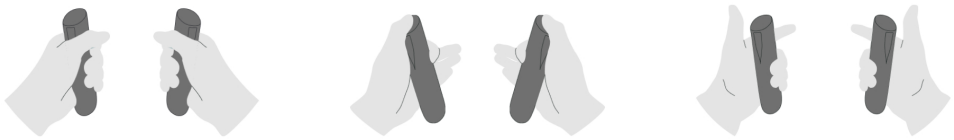
1.1.2 1.2. Sensor Data

1.1.2.1 1.2.1. Tactile Data

Tactile sensors can be found in the eteeControllers's main body, trackpad, and LED slider. If an etee tracker is attached, an additional proximity button can be accessed.



The finger tracking sensors are located in the eteeController's main body. For each finger, the eteeController detects two different levels of pressure, as well as touch and clicks. At the top of the device, there is a tactile trackpad. The eteeController estimates the XY position of a thumb on the trackpad and can also measure pressure and detect touch and click events. At the front of the main body, there is a LED light which also acts as a tactile sensor. If a tracker is connected to the eteeController, an additional proximity sensor can be used, located in the area around the tracker button. You can also retrieve the state of the system button (i.e. power button) located at the bottom of the device.



Multi-finger gesture detection is also provided by the device as grip, pinch and point gestures.

For more information on the retrievable values for each sensor or gesture, see *5.1.1. Tactile Data* or *5.1.2. Gestures*.

For more information on how to retrieve these values, see *4.2.1. Data Getters Functions*.

1.1.2.2 1.2.2. Inertial Measuring Units (IMU) Data

The IMU data consists of **acceleration, angular acceleration and magnetic flux density** data from the eteeControllers. This data is used by the library to estimate the spatial orientation of the eteeControllers through the calculation of **quaternions** and **Euler angles**.

For more information on the retrievable values from the IMU sensors, see *5.1.3. Inertial Measuring Units (IMU)*.

For more information on how to retrieve these values, see *4.3. How IMU Data Is Processed*.

1.1.2.3 1.2.3. Device and Battery State Data

Device state data includes information such as eteeController **handedness** (i.e. right or left) and **battery data**.

For more information on the retrievable values from the IMU sensors, see *5.1.4. Device State* or *5.1.5. Battery State*.

1.2 2. Setup

On this page

1.2.1 2.1. Hardware Setup

If this is your first time using the eteeControllers, please visit the etee XR guide page for more information on hardware setup. To interface with the etee Python API library, you will need a minimum of one eteeDongle and one eteeController.

Tip: It is also recommended to download the official etee Connect application, available in the Steam store. This application allows the user to visualize real-time sensor data from the eteeControllers, as well as update the devices' firmware or pair them, among many other features.

1.2.1.1 2.1.1. Controllers and eteeDongle Connection

Before getting started with the etee Python API, check the connection between your eteeControllers and eteeDongle.

First, **plug your eteeDongle** to your laptop or PC; when connected correctly, the LED indicator in the eteeDongle should consistently blink blue. If not, try re-plugging the eteeDongle.

Afterwards, **turn on your eteeController(s)**. Now, the eteeDongle should blink pink once if there is one eteeController connected to the eteeDongle, or twice if there are two eteeControllers connected. If the eteeDongle shows no connection with the eteeControllers, try pairing the devices again through the official *etee Connect* app, with the '*Settings* → *Pair Devices*' feature. You should also check that the eteeControllers and eteeDongle have compatible firmware versions.

1.2.1.2 2.1.2. Firmware Versions

The device firmware versions required for this API are:

- **eteeDongle Firmware**: 1.1.7 or higher.
- **eteeController(s) Firmware**: 1.3.2 or higher.

Firmware for etee devices can be updated through the official *etee Connect* app, under the '*Settings* → *Firmware*' Section.

1.2.2 2.2. Python Package Setup

The etee Python API requires Python 3.8 or higher. We highly recommend setting a virtual environment for your project, and installing this package within the environment.

Tip: **Note:** You will need the pip package to run the installation. If you do not have Python or pip installed, follow the Python guide on package installation.

1.2.2.1 2.2.1. Install from Github

If you wish to install the API package, see our step-by-step guides to install this package using the Github repository.

1.3 3. Using etee Python API

1.3.1 3.1. Quickstart

etee driver automatically detects an eteeDongle and connects to it through the EteeController class' **connect()** method. To start the eteeController data stream, call the class **start_data()**, followed by the **run()** method to initiate the data loop. While the data loop is running, the etee driver will read the eteeController serial data, parse it and store the results in its internal buffer.

```
# Import the etee API library
from etee import EteeController

# Instantiate the driver class
etee = EteeController()

etee.connect()     # Connect the eteeDongle to the driver
etee.start_data()  # Request eteeControllers to start data stream
etee.run()         # Start data loop
```

Once the driver is running, **data getter** functions can be called to get etee data.

```
while True:
   # Data getter functions
   left_trackpad_x = etee.get_trackpad_x('left')
   left_trackpad_y = etee.get_trackpad_y('left')
```

(continues on next page)

(continued from previous page)

```
    # Print the values
    print(f"Finger touch identified at position [{left_trackpad_x}, {left_trackpad_y}] in␣
↪the left eteeController.")
    time.sleep(0.01)
```

For more information, see: *4.2.1. Data Getters Functions*.

Alternatively, it may be preferable to retrieve data through an **event-based method**, where data processing functions can be added as callbacks to data events.

```
# Function to be added as callback
def process_left_index_pressure():
    left_index_pull = etee.get_index_pull('left')
    left_index_force = etee.get_index_force('left')
    print(f"The pressure of the left index finger is: pull = {left_index_pull}, force =
↪{left_index_force}.")

# Connect callback to data reception events
etee.left_hand_received.connect(process_left_index_pressure)
```

After this, when the driver starts running, the *process_left_index_pressure()* function will run every time a left-hand data packet is received by the driver.

The API library defines a few events, such as data receiving events and device connection/disconnection events, among others. A callback function can be connected to any of these events. For more information, see: *4.4. Event-Based Programming*.

1.3.2 3.2. Methods and Class Constructors

You can find the full reference of available methods and classes under the *6. API Reference* section.

1.4 4. About the API Functionalities

On this page

1.4.1 4.1. Creating Connections and Streaming Data

Serial connection between the driver and the eteeDongle is established by calling the **connect()** function. This function automatically detects the COM port numbers of available eteeDongles through a VID filtering system and creates a connection to the first instance. Connection to specific ports can also be established by calling the **connect_port()** function instead; however, it is recommended to use the default connect() function.

The **start_data()** function sends a serial command instructing the eteeControllers to start streaming sensor data.

The **run()** functions starts a data loop in a separate thread. The data loop reads serial data, parses it and stores it in an internal buffer. The data loop also listens to serial and data events and manages event callback functions.

```
etee = EteeController()
etee.connect()
etee.start_data()
etee.run()
```

To stop the etee driver, run the **stop()** function. To command the eteeControllers to stop streaming data, call the **stop_data()** function. To close the connection between the eteeDongle and the eteeControllers, run the **disconnect()** function.

```
etee.stop()
etee.stop_data()
etee.disconnect()
```

1.4.2 4.2. Retrieving Device Inputs

There are two ways to retrieve data from the eteeControllers, either using data getter functions or through event-based programming.

1.4.2.1 4.2.1. Data Getters Functions

Once the eteeController data stream is opened and the data loop is running, etee data is accessible through two methods: data getters and event-based methods.

When called, **data getter** functions retrieve the last data available in the internal buffer.

There are general-use data getters and specific-use data getters. For the general-use getters, the keys for the data and/or device to be retrieved need to be passed as arguments. In the case of the specific-use ones, these data getter functions retrieve specific tactile, gestures, IMU or device state values.

```
# General-use data getters
right_index_pull = etee.get_data("right", "index_pull")  # Option 1: get_data(dev, w)
right_index_pull = etee.get_right("index_pull")          # Option 2: get_right(w) or get_
↪left(w)

# Specific-use data getters
right_index_pull = etee.get_index_pull("right")
```

For the full list of input keys for the general-use getters, see *5.2. Data Packet Structure*.

For the full list of available data getters, particularly the specific-use getter functions, see: *Data getters*.

1.4.2.2 4.2.2. Event-Based Methods

The data getter functions can be specified as callbacks to be connected to specific events. For instance, we can connect functions to process left finger data only when data packets corresponding to the left eteeController are received. To learn more, see: *4.4. Event-Based Programming*.

1.4.3 4.3. How IMU Data Is Processed

The serial data includes raw IMU data which consist of accelerometer, gyroscope and magnetometer data. Data getter methods are available for IMU data retrieval, as defined in: *Data getters*. If the corresponding IMU data exists (i.e. is different from None), the data getters will return a list containing their x, y and z-axis values.

```
# Data getter for accelerometer
left_accel = etee.get_accel("left")
left_accel_x = left_accel[0]

# Data getters for gyroscope
left_gyro = etee.get_gyro("left")
left_gyro_y = left_gyro[1]

# Data getters for magnetometer
left_mag = etee.get_mag("left")
left_mag_z = left_mag[2]
```

The API library also provides methods to retrieve orientation data in the form of quaternions. The quaternion is computed during each data loop.

```
# Data getters for quaternions
quaternion_left = etee.get_quaternion("left")
quaternion_right = etee.get_quaternion("right")
```

Before using quaternions, **update_imu()** should be called. This function retrieves the IMU calibration data stored in the eteeControllers and uses it to compute the quaternions. Without calling this function, the calculated quaternions will not use the latest IMU calibration data. The IMU can be re-calibrated using the eteeConnect software, which will modify the internal IMU calibration parameters.

For orientation data, the user can also choose to use **absolute** or **relative orientation**. The former uses magnetometer data, while the latter does not. By default, absolute orientation is enabled.

```
# Absolute orientation is the default, but it can also be set as shown below
# This will use the magnetometer data.
etee.enable_absolute_imu(True)

# To disable the absolute orientation and switch to relative orientation
# This will NOT use the magnetometer data.
etee.enable_absolute_imu(False)
```

1.4.4 4.4. Event-Based Programming

The etee API library supports event-based programming. Several serial and data events are defined. Each event can be connected to callback functions which will be triggered when the event is identified within the data loop.

A callback function can be connected to an event by calling the **connect()** method.

The data events **left_hand_received** and **right_hand_received** occur respectively when data packets from the left or the right eteeController are received. In the code sample below, as long as the data loop is running, the processing functions will be called every time the corresponding hand data packet is received.

```
# Processing functions
def process_left_hand():
   left_fingers_pulls, left_fingers_forces = etee.get_device_finger_pressures("left")
   print(f"The pressure of the left pinky finger is: pull = {left_fingers_pulls[4]},␣
↪force = {left_fingers_forces[4]}.")

def process_right_hand():
   right_fingers_pulls, right_fingers_forces = etee.get_device_finger_pressures("right")
   print(f"The pressure of the right thumb finger is: pull = {right_fingers_pulls[4]},␣
↪force = {right_fingers_forces[4]}.")

# Connect the processing functions as callbacks for the events
etee.left_hand_received.connect(process_left_hand)
etee.right_hand_received.connect(process_right_hand)
```

Another example of data events are **left_hand_lost** and **right_hand_lost**. These events are triggered when the signal is lost from the left and right eteeController, respectively. Signal loss is defined as no data having been received from the eteeControllers in the last 0.5 seconds.

```
# Functions to be invoked
def on_left_hand_lost():
        print(f"The left data was lost.")

def on_right_hand_lost():
       print(f"The right data was lost.")

# Connect the functions as callbacks for the events
etee.left_hand_lost.connect(on_left_hand_lost)
etee.right_hand_lost.connect(on_right_hand_lost)
```

An example of a serial event is **dongle_disconnected**, which occurs when the serial connection to the eteeDongle is disrupted.

```
# Function to be invoked
def on_dongle_disconnect():
        print(f"The eteeDongle has been disconnected.")

# Connect the function as a callback for the event
etee.dongle_disconnected.connect(on_dongle_disconnect)
```

For the full API library reference for data and serial events, see: *Data getters*.

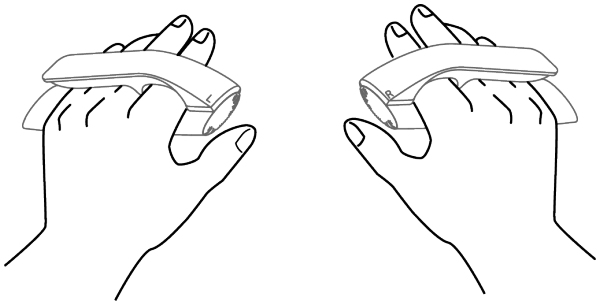
1.4.5 4.6. Resetting Sensor Baselines

If the finger curl data or other sensor data seem wrong, the easiest way to fix this is by triggering a quick calibration.

Quick calibration resets the electrodes signal baseline. For instance, if your real index finger is fully open but your virtual one appears to be flexed (i.e. index_pull is 55, instead of 0), a quick calibration will fix the issue (i.e. index_pull will be reset to 0).

Steps for quick calibration:

1. Please, ensure that your or the user's **fingers are fully stretched out**, as illustrated below.



2. **Trigger quick calibration** by using API commands or the physical button.

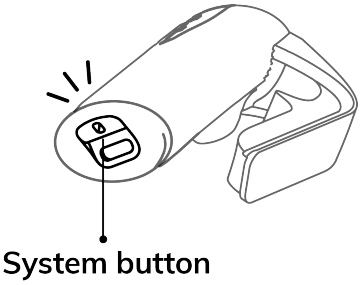
- **API commands**: You can send a quick calibration command to both or a single eteeController by writing the commands through serial communication to the eteeDongle.

```
# Quick calibrate all devices connected to eteeDongle
response = self.driver.send_command(b"BP+AB\r\n")

# Quick calibrate the right eteeController if connected to eteeDongle
response = self.driver.send_command(b"BR+AB\r\n")

# The response returns 'OK' when the command is successfully sent
print(response)
```

- **Physical button**: You can also trigger quick calibration by double-clicking the system button of the eteeController that you wish to reset the baselines.



3. This process is **instantaneous** and does not require any waiting time.

Note: This is **not the same as the full/advanced calibration** that takes place in eteeConnect.

We won't be providing any commands for this as it's a complicated step by step process. However, you should only need to run a full/advanced calibration if a different person is using the eteeControllers. In all other cases, a quick calibration will suffice.

1.5 5. About the eteeDongle Serial Communication

Data received from an eteeDongle can be either an etee data packet or lines of text written as a response to a user command or a hardware/firmware event. A data packet ends with a sequence 0xFFFF, while lines of text end with \r\n. These sequences should be used as delimiters when reading data from an etee serial.

The user can communicate with the eteeControllers through commands. An example of etee commands is starting and stopping data streams.

On this page

1.5.1 5.1. etee Packet Elements

Each etee packet corresponds to one reading of data from left or right eteeController. One packet of data is 44 bytes long in total. This includes 42 bytes for eteeController data and 2 delimiter bytes at the end of the packet. The packet delimiter is the sequence x0FFFF. A data packet contains tactile data, IMU measurements and device state information.

The following list describes the etee packet with parts grouped logically. The table below shows how and where the data appears in the packet. For more details on the sensor locations within the device, see *1. Overview*.

1.5.1.1 5.1.1. Tactile Data

Finger data – For each of **thumb**, **index**, **middle**, **ring**, **pinky**:

- **{finger}_pull (uint)** – First analog range corresponding to softer pressure of a finger, usually indicative of finger flexion.
- **{finger}_force (uint)** – Second analog range corresponding to harder pressure of a finger, usually indicating finger squeeze.
- **{finger}_touched (bool)** – Boolean indicating a light touch of a finger on the eteeController.
- **{finger}_clicked (bool)** – Boolean indicating a harder press of a finger on the eteeController.

Trackpad:

- **trackpad_x (uint)** – X (horizontal) coordinate of the estimated thumb position on a trackpad.

- **trackpad_y (uint)** – Y (vertical) coordinate of the estimated thumb position on a trackpad.
- **trackpad_pull (uint)** – First analog pressure value corresponding to light pressure of a thumb on the trackpad.
- **trackpad_force (uint)** – Second analog pressure value corresponding to hard press of a thumb on the trackpad.
- **trackpad_touched (bool)** – Boolean indicating a light touch of a thumb on the trackpad.
- **trackpad_clicked (bool)** – Boolean indicating a harder press of a thumb on the trackpad.

Slider (Integrated under the LED light):

- **slider_value (uint)** – Y (vertical) coordinate of the estimated touch position on the slider.
- **slider_touched (bool)** – Boolean indicating that the slider (any part) was touched.
- **slider_up_touched (bool)** – Boolean indicating that the upper part of the slider was touched.
- **slider_down_touched (bool)** – Boolean indicating that the lower part of the slider was touched.

Tracker – Optional: If a tracker is attached to a eteeController:

- **tracker_on (bool)** – Boolean indicating that a tracker is connected to the eteeController.
- **proximity_value (uint)** – Analog value corresponding to proximity to the tracker's sensor.
- **proximity_touched (bool)** – Boolean indicating that a tracker's proximity sensor is at touch level.
- **proximity_clicked (bool)** – Boolean indicating that a tracker's proximity sensor is at click level.

1.5.1.2 5.1.2. Gestures

Grip Gesture – Gesture triggered when squeezing all fingers around the eteeController. A grip gesture is defined by:

- **grip_pull (uint)** – First analog pressure range corresponding to light grip.
- **grip_force (uint)** – Second analog pressure range corresponding to hard grip.
- **grip_touched (bool)** – Light grip around the eteeController.
- **grip_clicked (bool)** – Grip gesture triggered.

Pinch Gesture – Gesture with trackpad/thumb and index fingers closed on the eteeController. There are 2 variations of pinch: **trackpad** (pinch with trackpad and index finger) and **thumbfinger** (pinch with thumb finger and index finger). A pinch gesture is defined by:

- **pinch_{variation}_pull (uint)** – Analog pressure range for the pinch gesture.
- **pinch_{variation}_clicked (bool)** – Pinch gesture variation triggered.

Point Gesture – Gesture with index finger extended and the others closed on the eteeController. There are 2 variations of point: **exclude_trackpad** (trackpad must not be touched) and **independent** (trackpad can be used alongside the gesture). A point gesture is defined by:

- **point_{variation}_clicked (bool)** – Point gesture variation triggered.

1.5.1.3 5.1.3. Inertial Measuring Units (IMU)

IMU Data – For each of **x**, **y** and **z** components:

- **accel_{component} (int)** – Acceleration data from the accelerometer.
- **gyro_{component} (int)** – Angular acceleration data from the gyroscope.
- **mag_{component} (int)** – Magnetic flux density data from the magnetometer.

1.5.1.4 5.1.4. Device State

- **is_right_hand (bool)** – Boolean indicating whether the data packet comes from the right (1) or the left (0) eteeController.
- **system_button (bool)** – System button pressed.

1.5.1.5 5.1.5. Battery State

- **battery_level (uint)** – Battery fuel gauge level.
- **battery_charging (bool)** – Boolean indicating if the device is plugged in and charging (true) or not (false).
- **battery_charging_complete (bool)** – Boolean indicating if the device has finished charging (true) or not (false).

1.5.2 5.2. Data Packet Structure

The location of the parameters in the data packet can be found in the table below.

Note: When parsing the data, the parameter location within the data packet is given as follows: *bit_offset (byte, bit), length_in_bits*.

Table 1: etee Packet

Location	Name	Type	Range
0 (0, 0), 1	system_button	bool	0 – 1
1 (0, 1), 1	trackpad_clicked	bool	0 – 1
2 (0, 2), 1	trackpad_touched	bool	0 – 1
3 (0, 3), 1	thumb_clicked	bool	0 – 1
4 (0, 4), 1	index_clicked	bool	0 – 1
5 (0, 5), 1	middle_clicked	bool	0 – 1
6 (0, 6), 1	ring_clicked	bool	0 – 1
7 (0, 7), 1	pinky_clicked	bool	0 – 1
8 (1, 0), 1	thumb_touched	bool	0 – 1
9 (1, 1), 7	thumb_pull	uint	0 – 126
16 (2, 0), 1	index_touched	bool	0 – 1
17 (2, 1), 7	index_pull	uint	0 – 126
24 (3, 0), 1	middle_touched	bool	0 – 1
25 (3, 1), 7	middle_pull	uint	0 – 126
32 (4, 0), 1	ring_touched	bool	0 – 1
33 (4, 1), 7	ring_pull	uint	0 – 126
40 (5, 0), 1	pinky_touched	bool	0 – 1

continues on next page

Table 1 – continued from previous page

Location	Name	Type	Range
41 (5, 1), 7	pinky_pull	uint	0 – 126
48 (6, 0), 8	trackpad_x	int	0 – 255
56 (7, 0), 8	trackpad_y	uint	0 – 255
64 (8, 0), 1	proximity_touch	bool	0 – 1
65 (8, 1), 7	proximity_value	uint	0 – 126
72 (9, 0), 1	slider_touched	bool	0 – 1
72 (9, 1), 7	slider_value	uint	0 – 126
80 (10, 0), 1	grip_touched	bool	0 – 1
80 (10, 1), 7	grip_pull	uint	0 – 126
88 (11, 0), 1	grip_clicked	bool	0 – 1
89 (11, 1), 1	proximity_clicked	bool	0 – 1
90 (11, 2), 1	tracker_on	bool	0 – 1
91 (11, 3), 1	is_right_hand	bool	0 – 1
92 (11, 4), 1	battery_charging	bool	0 – 1
93 (11, 5), 1	slider_up_touched	bool	0 – 1
94 (11, 6), 1	slider_down_touched	bool	0 – 1
96 (12, 0), 1	battery_charge_complete	bool	0 – 1
97 (12, 1), 7	battery_level	uint	0 – 100
104 (13, 0), 1	point_exclude_trackpad_clicked	bool	0 – 1
105 (13, 1), 7	trackpad_pull	uint	0 – 126
112 (14, 0), 1	point_independent_clicked	bool	0 – 1
113 (14, 1), 7	grip_force	uint	0 – 126
120 (15, 0), 1	pinch_trackpad_clicked	bool	0 – 1
121 (15, 1), 7	pinch_trackpad_pull	uint	0 – 126
128 (16, 0), 1	pinch_thumbfinger_clicked	bool	0 – 1
129 (16, 1), 7	pinch_thumbfinger_pull	uint	0 – 126
137 (17, 1), 7	trackpad_force	uint	0 – 126
145 (18, 1), 7	thumb_force	uint	0 – 126
153 (19, 1), 7	index_force	uint	0 – 126
161 (20, 1), 7	middle_force	uint	0 – 126
169 (21, 1), 7	ring_force	uint	0 – 126
177 (22, 1), 7	pinky_force	uint	0 – 126
184 (23, 0), 16	accel_x	int	-32768 – 32767
200 (25, 0), 16	accel_y	int	-32768 – 32767
216 (27, 0), 16	accel_z	int	-32768 – 32767
232 (29, 0), 16	mag_x	int	-32768 – 32767
248 (31, 0), 16	mag_y	int	-32768 – 32767
264 (33, 0), 16	mag_z	int	-32768 – 32767
280 (35, 0), 16	gyro_x	int	-32768 – 32767
296 (37, 0), 16	gyro_y	int	-32768 – 32767
312 (39, 0), 16	gyro_z	int	-32768 – 32767
328 (41, 0), 8	unused_byte	n/a	n/a

The location of specific values in the data packet is defined in a YAML file inside of the etee Python API package.

1.5.3 5.3. Commands

Commands are sent to etee in the format `PF + CM [=PM]`. Where **PF** is a prefix of the command which can be one of the following

- **BP** – command is directed to both left and right eteeControllers.
- **LP** – command is directed to the left etee only.
- **RP** – command is directed to the right etee only.

CM is a command name. The **optional PM** are the command parameters. The table below lists commands supported by the API library.

Table 2: etee Commands

Command	Description	Response
AG	Start etee data stream	OK
AS	Stop etee data stream	OK
RB	Quick calibration - Resets the signal baselines for all sensors. See *4.6. Resetting Sensor Baselines*.	OK

For example, to start a data stream in the left controller, the command would be: `LP+AG`.

1.5.4 5.4. Event Prints

Event prints are lines of text written to serial as a response to hardware/firmware events. The API library provides the following event prints.

Table 3: Event Prints

Print	Event
L connection complete	Left eteeController detected
R connection complete	Right eteeController detected
L disconnected	Left eteeController disconnected
R disconnected	Right eteeController disconnected

1.6 6. API Reference

Tip: Use the **search tool** in the top right of this page to quick search methods or events

You can find below a list of methods and class constructors available from the etee Python API package.

1.6.1 EteeController Class

On this page

Here you can find the list of methods from the EteeController class, which acts as a driver enabling communication between the eteeControllers and a computer through an eteeDongle.

class etee.driver_eteecontroller.**EteeController**

This class manages the communication between the driver and the eteeControllers, through the eteeDongle. It also handles the data loop which retrieves and stores the eteeControllers data into an internal buffer, allowing for their retrieval through its class methods.

1.6.1.1 Connections

The following methods manage the connection between the eteeDongle and the driver, and activate/deactivate the eteeControllers data streams.

EteeController.**connect**()

Establish serial connection to an etee dongle. This function automatically detects etee dongles connected to a COM port and connects to the first available one.

EteeController.**connect_port**(*port=None*)

Attempt to establish serial connection to an etee dongle port. If a COM port argument is provided, connection is attempted with the specified port. If the port argument is None, the driver automatically detects any COM ports with an etee dongle and connects to the first available one. Default port value is None.

Parameters
port (*str or None*) – etee dongle COM port.

Returns
Success flag - True if the connection is successful, False if otherwise

Return type
bool

EteeController.**disconnect**()

Close serial connection to etee dongle.

Returns
Success flag - True if the connection was closed successfully, False if otherwise

Return type
bool

EteeController.**run**()

Initiates the data loop in a separate thread. The data loop reads serial data, parses it and stores it in an internal buffer. The data loop also listens to serial and data events and manages event callback functions.

EteeController.**stop**()

Stops the data loop.

EteeController.**start_data**()

Sends command to the etee controller to start the data stream.

EteeController.**stop_data**()

Sends command to the etee controller to stop the data stream.

EteeController.**get_available_etee_ports**()

Get all available etee dongle COM ports. Other devices are automatically filtered out through a VID and PID filtering method.

Returns
List of COM port names with etee dongles connected.

Return type
list[str]

1.6.1.2 Events

A callback function can be connected to the following events through the *etee.driver_eteecontroller.EteeControllerEvent.connect()* method in the *etee.driver_eteecontroller.EteeControllerEvent* class.

EteeController.**left_hand_received**

Event for receiving left controller data.

Type
Event

EteeController.**right_hand_received**

Event for receiving right controller data.

Type
Event

EteeController.**hand_received**

Event for receiving data from any controller.

Type
Event

EteeController.**left_hand_lost**

Event for losing left controller connection. Occurs when data is not received for more than 0.5 seconds.

Type
Event

EteeController.**right_hand_lost**

Event for losing right controller connection. Occurs when data is not received for more than 0.5 seconds.

Type
Event

EteeController.**data_lost**

Event for losing data from both controllers. Occurs when data is not received for more than 0.5 seconds.

Type
Event

EteeController.**left_connected**

Event for left controller connection detected by the dongle.

Type
Event

EteeController.**right_connected**

Event for right controller connection detected by the dongle.

Type
Event

EteeController.**left_disconnected**

Event for left controller disconnection from the dongle.

Type
Event

EteeController.**right_disconnected**

Event for right controller disconnection from the dongle.

Type
Event

EteeController.**dongle_disconnected**

Event for dongle disconnection.

Type
Event

1.6.1.3 IMU Processing

There are also several methods that can be used to process IMU data at the start of or during your application.

Enable Absolute / Relative Orientation

The orientation can also be set to absolute (i.e. magnetometer data used) or relative (i.e. no magnetometer data used) through the following method:

EteeController.**absolute_imu_enabled**(*on*)

Enables or disables absolute orientation. Absolute orientation uses data from the accelerometer, gyroscope and magnetometer sensors for quaternion calculations. If disabled, the default mode will be enabled, which uses relative orientation, calculated only though the accelerometer and gyroscope data.

Parameters
on (*bool*) – True to switch to absolute orientation, False for relative orientation.

Set Offsets

Gyroscope and magnetometer offsets can be easily set at any time using the following methods:

EteeController.**update_gyro_offset_left**()

Retrieves the gyroscope calibration parameters saved on the left etee controller, and updates the calibration offsets in the driver model.

EteeController.**update_gyro_offset_right**()

Retrieves the gyroscope calibration parameters saved on the right etee controller, and updates the calibration offsets in the driver model.

EteeController.**update_mag_offset_left**()

Retrieves the magnetometer calibration parameters saved on the left etee controller, and updates the calibration offsets in the driver model.

EteeController.**update_mag_offset_right**()

Retrieves the magnetometer calibration parameters saved on the right etee controller, and updates the calibration offsets in the driver model.

EteeController.**update_imu_offsets**()

Retrieves the gyroscope and magnetometer calibration parameters from both controllers, and updates the calibration offsets in the driver model.

1.6.1.4 Data getters

When called, data getter functions allow for the retrieval of device data from the driver's internal buffer. You can find the full list of data getter methods from the EteeController class below.

General-Use Data Getters

The same general-use data getter can be used to retrieve a variety of eteeController values through a single method. For these methods, the keys for the data and/or device to be retrieved need to be passed as arguments.

Possible keys: see *5.2. Data Packet Structure*.

EteeController.**get_left**(*w*)

Get a key value in the current internal data buffer for the left device.

Parameters
w (*str*) – Key for the device data to be retrieved, as defined in the YAML file.

Returns
Left controller's value for the key provided.

EteeController.**get_right**(*w*)

Get a key value in the current internal data buffer for the right device.

Parameters
w (*str*) – Key for the device data to be retrieved, as defined in the YAML file.

Returns
Right controller's value for the key provided.

EteeController.**get_data**(*dev*, *w*)

Get a key value in the current internal data buffer for the specified device (left or right).

Parameters

- **dev** (*str*) – Selected controller hand. Possible values: "left", "right".
- **w** (*str*) – Key for the device data to be retrieved, as defined in the YAML file.

Returns
Selected controller's value for the key provided.

Raises
ValueError – if the dev input is not "left" or "right"

Specific-Use Data Getters

Unlike the general-use getter methods, the specific-use methods retrieve specific tactile, gestures, IMU or device state values from a specified eteeController. You can find the full list of them below.

Hand/Controller Connection Status

Whether the eteeController devices are detected as connected.

`EteeController.all_hands_on()`

Check if both left and right controllers are connected.

Returns
Returns true if data has been recently received from both controllers.

Return type
bool

`EteeController.any_hand_on()`

Check if either left or right controller is connected.

Returns
Returns true if data has been recently received from any of the controller.

Return type
bool

`EteeController.left_hand_on()`

Check if the left controller is connected.

Returns
Returns true if data has been recently received from the left controller.

Return type
bool

`EteeController.right_hand_on()`

Check if the right controller is connected.

Returns
Returns true if data has been recently received from the right controller.

Return type
bool

Firmware Versions

Returns the eteeDongle or devices firmware version.

`EteeController.get_dongle_version()`

Retrieve the firmware version of the connected dongle.

Returns
Returns the dongle firmware version if a dongle is connected. If no dongle is connected, the firmware version value will be None.

Return type
str

EteeController.**get_etee_versions**()

Retrieve the firmware version from the connected controllers.

Returns
Returns the firmware versions of the connected controllers. If a controller is not connected, its firmware version value will be None.

Return type
list[str]

Finger Data

Thumb Finger Data

EteeController.**get_thumb_pull**(*dev*)

Returns the thumb finger pull value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Thumb finger pull pressure (i.e. first pressure range, corresponding to light touch) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_thumb_force**(*dev*)

Returns the thumb finger force value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Thumb finger force pressure (i.e. second pressure range, corresponding to squeeze levels) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_thumb_touched**(*dev*)

Returns the thumb finger touch value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s thumb finger is touched.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_thumb_clicked**(*dev*)

Returns the thumb finger click value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s thumb finger is clicked.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Index Finger Data

EteeController.**get_index_pull**(*dev*)

Returns the index finger pull value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Index finger pull pressure (i.e. first pressure range, corresponding to light touch) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_index_force**(*dev*)

Returns the index finger force value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Index finger force pressure (i.e. second pressure range, corresponding to squeeze levels) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_index_touched**(*dev*)

Returns the index finger touch value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller's index finger is touched.

Return type
bool

Raises
ValueError – if the dev input is not "left" or "right"

`EteeController.`**`get_index_clicked`**`(`*dev*`)`

Returns the index finger click value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
True if the selected controller's index finger is clicked.

Return type
bool

Raises
ValueError – if the dev input is not "left" or "right"

Middle Finger Data

`EteeController.`**`get_middle_pull`**`(`*dev*`)`

Returns the middle finger pull value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
Middle finger pull pressure (i.e. first pressure range, corresponding to light touch) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not "left" or "right"

`EteeController.`**`get_middle_force`**`(`*dev*`)`

Returns the middle finger force value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
Middle finger force pressure (i.e. second pressure range, corresponding to squeeze levels) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not "left" or "right"

EteeController.**get_middle_touched**(*dev*)

Returns the middle finger touch value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s middle finger is touched.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_middle_clicked**(*dev*)

Returns the middle finger click value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s middle finger is clicked.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Ring Finger Data

EteeController.**get_ring_pull**(*dev*)

Returns the ring finger pull value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Ring finger pull pressure (i.e. first pressure range, corresponding to light touch) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_ring_force**(*dev*)

Returns the ring finger force value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Ring finger force pressure (i.e. second pressure range, corresponding to squeeze levels) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_ring_touched**(*dev*)

Returns the ring finger touch value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s ring finger is touched.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_ring_clicked**(*dev*)

Returns the ring finger click value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s ring finger is clicked.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Pinky Finger Data

EteeController.**get_pinky_pull**(*dev*)

Returns the pinky finger pull value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Pinky finger pull pressure (i.e. first pressure range, corresponding to light touch) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_pinky_force**(*dev*)

Returns the pinky finger force value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Pinky finger force pressure (i.e. second pressure range, corresponding to squeeze levels) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_pinky_touched**(*dev*)

Returns the pinky finger touch value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s pinky finger is touched.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_pinky_clicked**(*dev*)

Returns the pinky finger click value for the selected device/controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected controller’s pinky finger is clicked.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Pressure from all fingers

EteeController.**get_device_finger_pressures**(*dev*)

Returns all the fingers pull and force pressure values.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Two arrays containing the pull and force pressures for the selected controller’s five finger sensors. The first element of each array will be the thumb, and the las the pinky. For example: fingers_pull[2] = index finger pull. Pull and force values range: 0-126. Base values: 0.

Return type
list[int], list[int]

Raises
ValueError – if the dev input is not “left” or “right”

Trackers Connection Status

EteeController.**get_tracker_connections**()

Checks if both eteeTrackers are connected to the controllers.

Returns
Returns True if both trackers are connected.

Return type
bool

EteeController.**get_tracker_connection**(*dev*)

Checks if the selected controller has an eteeTracker connected.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Returns True if the selected tracker is connected.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Trackers Data - Proximity Sensor

EteeController.**get_proximity**(*dev*)

Returns the proximity sensor analog value for the selected controller. This sensor is only available when an eteeTracker is connected. If disconnected, the value will always be 0, even when the sensor is interacted with.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Value of the selected tracker’s proximity sensor. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_proximity_touched**(*dev*)

Returns the proximity sensor touch value for the selected controller. This sensor is only available when an eteeTracker is connected. If disconnected, the value will always be false, even when the sensor is touched.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected tracker’s proximity sensor value is at touch level.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_proximity_clicked**(*dev*)

Returns the proximity sensor click value for the selected controller. This sensor is only available when an eteeTracker is connected. If disconnected, the value will always be false, even when the sensor is touched.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected tracker’s proximity sensor value is at click level.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Trackpad Data

EteeController.**get_trackpad_x**(*dev*)

Returns the trackpad x-axis position for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Trackpad X (horizontal) coordinate for the selected controller. Range: 0-255. If not touched, the value is 126.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_trackpad_y**(*dev*)

Returns the trackpad y-axis position for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Trackpad Y (vertical) coordinate for the selected controller. Range: 0-255. If not touched, the value is 126.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_trackpad_xy**(*dev*)

Returns the trackpad x-axis and y-axis positions for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Trackpad XY coordinates for the selected controller. Range for each axis coordinate: 0-255. If not touched, the value is 126.

Return type
list[int]

Raises
ValueError – if the dev input is not "left" or "right"

EteeController.**get_trackpad_pull**(*dev*)

Returns the trackpad pull pressure value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
Trackpad pull pressure (i.e. first pressure range, corresponding to light touch) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not "left" or "right"

EteeController.**get_trackpad_force**(*dev*)

Returns the trackpad force pressure value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
Trackpad force pressure (i.e. second pressure range, corresponding to hard press) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not "left" or "right"

EteeController.**get_trackpad_touched**(*dev*)

Returns the trackpad touch value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
True if the selected trackpad is touched.

Return type
bool

Raises
ValueError – if the dev input is not "left" or "right"

EteeController.**get_trackpad_clicked**(*dev*)

Returns the trackpad click value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
True if the selected trackpad is clicked.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

LED - Slider Data

EteeController.**get_slider_value**(*dev*)

Returns the slider positional value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
LED slider position, alongside its Y-axis (vertical), for the selected controller. Range: 0-126. If not touched, the slider value is 126.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_slider_touched**(*dev*)

Returns the slider touch value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected LED light is touched.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_slider_up_button**(*dev*)

Returns the slider UP button value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the upper part of selected LED is touched.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_slider_down_button**(*dev*)

Returns the slider DOWN button value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the lower part of selected LED is touched.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Gestures

Grip Gesture

EteeController.**get_grip_pull**(*dev*)

Returns the grip gesture’s pull pressure value for the selected controller. If the gesture is not performed, the pull value will be 0.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Grip gesture’s pull pressure (i.e. first pressure range, corresponding to light touch) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_grip_force**(*dev*)

Returns the grip gesture’s force pressure value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Grip gesture’s force pressure (i.e. second pressure range, corresponding to squeeze levels) for the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_grip_touched**(*dev*)

Returns the grip gesture’s touch value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the grip gesture reaches touch level in the selected controller.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_grip_clicked**(*dev*)

Returns the grip gesture’s click value for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the grip gesture reaches click level in the selected controller.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Pinch Gesture - Trackpad Variation

Pinch Variation: The standard pinch gesture is triggered by pressing the trackpad and index finger

EteeController.**get_pinch_trackpad_pull**(*dev*)

Returns the pull pressure value for the pinch with trackpad gesture in the selected controller. If the gesture is not performed, the pull value will be 0.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Pull pressure for the pinch gesture (trackpad variation) in the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_pinch_trackpad_clicked**(*dev*)

Returns the click value for the pinch with trackpad gesture in the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the pinch gesture (trackpad variation) reaches click level in the selected controller.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Pinch Gesture - Thumb Finger Variation

Pinch Variation: This alternative pinch gesture is triggered by pressing the thumb finger and index finger, instead of the trackpad as shown in the standard pinch.

`EteeController.get_pinch_thumbfinger_pull(dev)`

Returns the pull pressure value for the pinch with thumb finger gesture in the selected controller. If the gesture is not performed, the pull value will be 0.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
Pull pressure for the pinch gesture (thumb finger variation) in the selected controller. Range: 0-126. Base value: 0.

Return type
int

Raises
ValueError – if the dev input is not "left" or "right"

`EteeController.get_pinch_thumbfinger_clicked(dev)`

Returns the click value for the pinch with thumb finger gesture in the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
True if the pinch gesture (thumb finger variation) reaches click level in the selected controller. Range: 0-126. Base value: 0.

Return type
bool

Raises
ValueError – if the dev input is not "left" or "right"

Point Gesture - Exclude Trackpad

Point Variation: In the standard point gesture, the trackpad must not be touched.

`EteeController.get_point_excl_tp_clicked(dev)`

This is the alternative point gesture.

Returns the click value for the exclude-trackpad point (trackpad must not be touched) gesture in the selected controller. In this variation, if the user touches the trackpad while doing the point gesture, the gesture will be cancelled.

Parameters
dev (*str*) – Selected device hand. Possible values: "left", "right".

Returns
True if the exclude-trackpad point gesture variation (i.e. where the trackpad is not touched) is detected in the selected controller.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

Point Gesture - Independent

Point Variation: In this alternative point gesture, the trackpad can be used alongside this point gesture.

EteeController.**get_point_independent_clicked**(*dev*)

This is the main point gesture used in VR and XBOX-controller based games.

Returns the click value for the independent point (trackpad can be touched) gesture in the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the independent point gesture variation is detected in the selected controller. In this variation, the trackpad can be used alongside the point gesture.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

IMU and Quaternions

This API provides estimations of device three-dimensional orientation in the form of **quaternions** and **Euler angles**.

EteeController.**get_quaternion**()

Returns the quaternion for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Rotation quaternion for the selected controller.

Return type
list[int]

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_euler**()

Returns the euler angles for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Euler angles (roll, pitch, yaw) for the selected controller.

Return type
list[int]

Raises
ValueError – if the dev input is not “left” or “right”

Raw data from the 9-axis IMU can also be retrieved using the following methods:

EteeController.**get_accel**(*dev*)

Returns the accelerometer values for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Acceleration vector for the selected controller.

Return type
list[int]

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_gyro**(*dev*)

Returns the gyroscope values for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Angular acceleration vector for the selected controller.

Return type
list[int]

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_mag**(*dev*)

Returns the magnetometer values for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Magnetic flux density vector for the selected controller.

Return type
list[int]

Raises
ValueError – if the dev input is not “left” or “right”

Battery

EteeController.**get_battery_level**(*dev*)

Returns the battery level for the selected controller.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
Battery fuel gauge level for the selected controller. Range: 0-100.

Return type
int

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_charging_in_progress_status**(*dev*)

Checks if the selected controller is charging.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected battery is charging.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

EteeController.**get_charging_complete_status**(*dev*)

Checks if the selected controller has finished charging (battery level is 100%).

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected battery charging has been completed.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

System/Power Button

EteeController.**get_system_button_pressed**(*dev*)

Checks if the system button is pressed.

Parameters
dev (*str*) – Selected device hand. Possible values: “left”, “right”.

Returns
True if the selected system button is pressed.

Return type
bool

Raises
ValueError – if the dev input is not “left” or “right”

1.6.2 Event Class

Here you can find the list of methods from the Event class, which allow callback functions to be connected or disconnected from etee events, defined in the *etee.driver_eteecontroller.EteeController* class.

class etee.driver_eteecontroller.**EteeControllerEvent**

This class manages eteeController driver events by allowing callback functions to be connected or disconnected from them.

connect(*callback*)

Connect a callback function to the event.

Parameters
callback (*callable*) – Callback function.

disconnect(*callback*)

Remove a function from the event callbacks.

Parameters
callback (*callable*) – Callback function.

emit()

Emit event.

1.7 Go to: Github Repository

To visit our Github repository for the etee Python API, please click here: https://github.com/eteeXR/etee-Python-API

INDEX