1. About

1.1. Introduction

This API allows developers to easily create applications that interface with the etee finger-tracking controllers. It enables seamless device data reading and communication, streamlining the process of integrating the controllers into your software development workflow.

The package contains

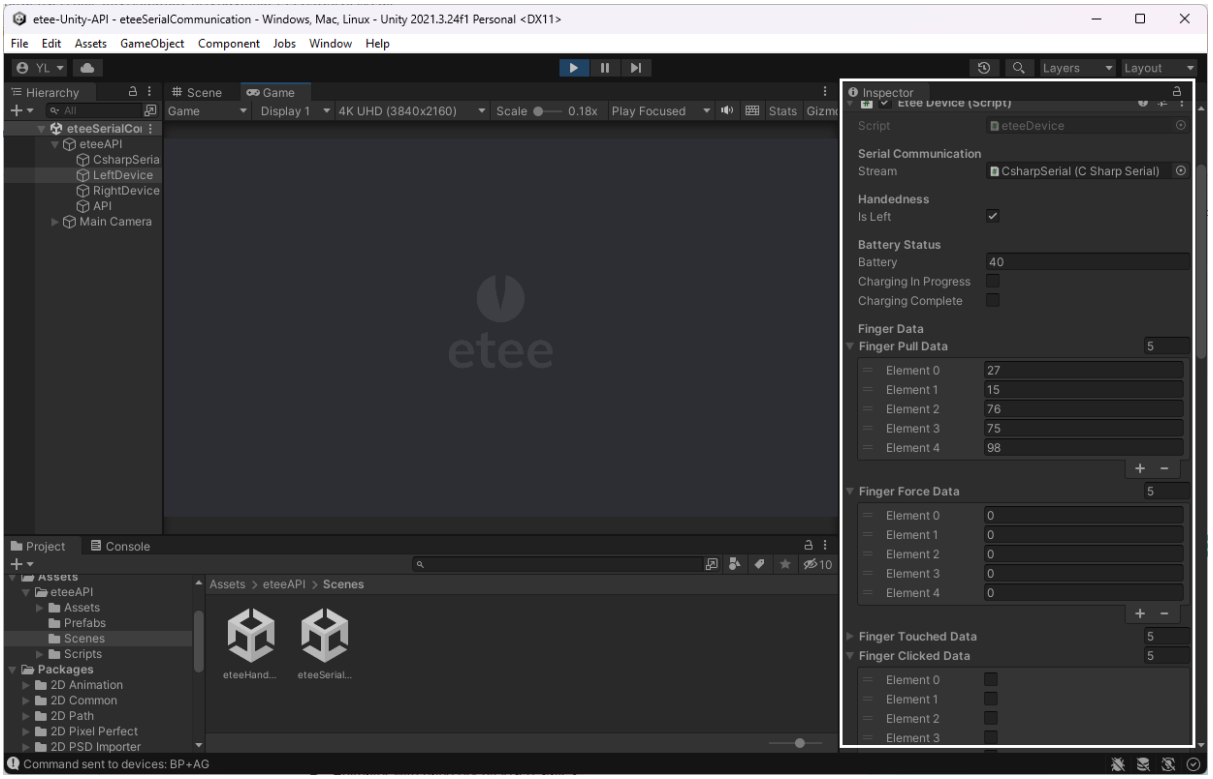
- Pkg - etee API
- Scenes (one for basic device communication and one for hand movement visualisation)
- Assets required for the basic set-up

The API contains four main scripts:

- **CsharpSerial.cs** - Retrieves device and port statuses. It also initialises device connection and data streaming commands.
- **EteeDevice.cs** - Gathers values from the device data packet.
- **EteeAPI.cs** - Call and retrieves values from the API
- **AHRS.cs** - Processes rotation information

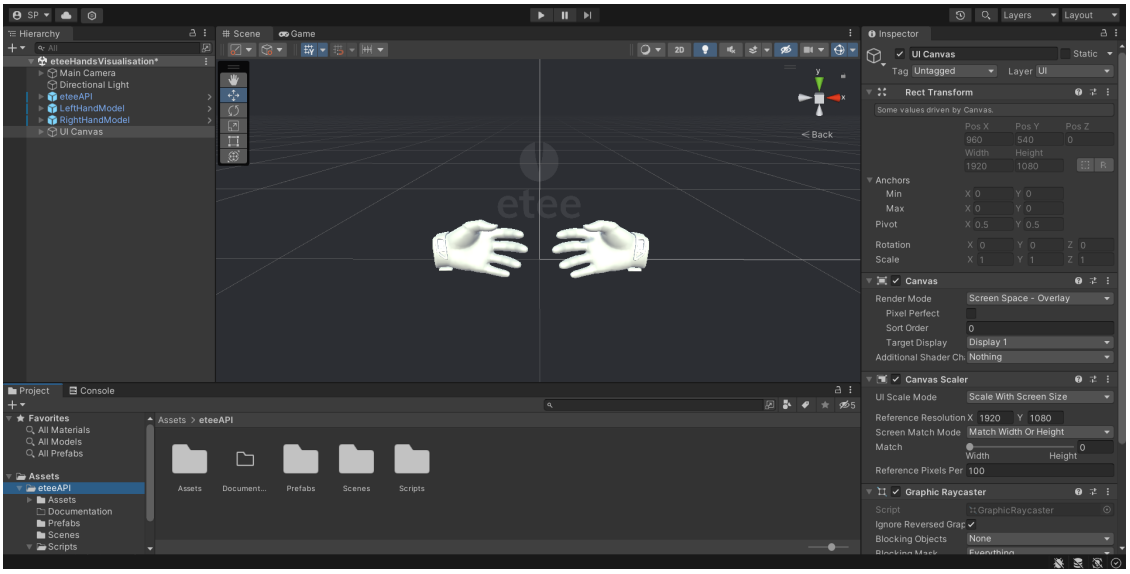
1.2. Scenes

- eteeSerialCommunication - This scene only contains the etee API prefab for developers to interact with.



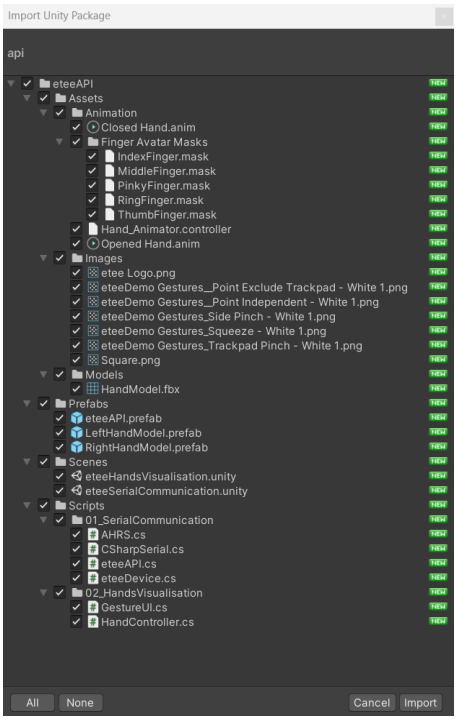
- eteeHandsVisualisation - This scene contains a visual rendition of the user's hands that mimic their gestures using the etee controller. Included in this scene are the following:
    - One pair of hand prefabs
        - Animator with hand open/closed states
        - 3D hand model
        - Hand controller script (retrieves API values for animation)

    - Simple Gesture UI to visualise which gesture is being performed on each hand:
        - Gesture Icons
        - Gesture UI script (retrieves API gesture states and controls the icons shown)



1.3. Getting Started

Begin by importing the etee API package into your Unity project. To do this, go to **Window > Package Manager**, select **My Assets**, find the etee API package, and click **Import**. Make sure everything is selected when importing to avoid dependency issues within the API.



With the package imported make sure you switch to the correct API Compatibility Level within **Edit > Project Settings > Player > Other Settings > API Compatibility Level**. If you are developing in Unity version 2021 select **.NET Framework**, if you are using an older version of Unity you should use **.NET 4.x** instead.

2. Hardware Setup

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

2.1. Controllers and eteeDongle Connection

Before starting with the etee API, check the connection between your eteeControllers and eteeDongle.

First, **plug your eteeDongle** into 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, re-plug the dongle. You should also check that the eteeControllers and eteeDongle have compatible firmware versions.

2.2. Firmware Versions

The device firmware versions required for this API are:

- **eteeDongle Firmware**: 1.1.7 or higher.
- **eteeController(s) Firmware**: 1.3.6 or higher.

Firmware for etee devices can be updated through the official *etee Connect* app, under the ‘Settings ‣ Firmware’ Section.

3. etee Packet Elements

Each etee packet corresponds to one reading of data from the 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.

3.1.1. Tactile Data

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

- **{finger}PullValue (float)** – First analogue range corresponding to softer pressure of a finger, usually indicative of finger flexion.
- **{finger}ForceValue (float)** – Second analogue 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:

- **trackpadCoordinates (Vector3)** – X (horizontal) coordinate of the estimated thumb position on a trackpad. Y (vertical) coordinate of the estimated thumb position on a trackpad.
- **trackpadPressures (Tuple<float, float>)** – First analogue pressure value corresponding to the light pressure of a thumb on the trackpad. The second analogue pressure value corresponds to the hard press of a thumb on the trackpad.
- **trackpadTouched (bool)** – Boolean indicating a light touch of a thumb on the trackpad.
- **trackpadClicked (bool)** – Boolean indicating a harder thumb press on the trackpad.

Slider (Integrated under the LED light):

- **sliderValue (float)** – Y (vertical) coordinate of the estimated touch position on the slider.
- **sliderButton (bool)** – Boolean indicating that the slider (any part) was touched.
- **sliderUpButton (bool)** – Boolean indicating that the upper part of the slider was touched.
- **sliderDownButton (bool)** – Boolean indicating that the lower part of the slider was touched.

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

- **trackerConnected (bool)** – Boolean indicating that a tracker is connected to the eteeController.
- **proxValue (float)** – Analog value corresponding to proximity to the tracker's sensor.
- **proxTouched (bool)** – Boolean indicating that a tracker's proximity sensor is at touch level.
- **proxClicked (bool)** – Boolean indicating that a tracker's proximity sensor is at click level.

3.1.2. Gestures

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

- **squeeze** - Grip gesture triggered.
- **gripPressures(Tuple<float,float>)** – First analogue pressure range corresponds to light grip. The second analogue pressure range corresponds to a hard grip.
- **gripTouch (bool)** – Light grip around the eteeController.
- **gripClick (bool)** – Hard grip around the eteeController

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}Analog (float)** – Analog pressure range for the pinch gesture.
- **pinch{variation} (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}Analog (float)** - Analog pressure range for point gesture.
- **point{variation} (bool)** – Point gesture variation triggered.

3.1.3. Inertial Measuring Units (IMU)

IMU Data – For each x, y and z component:

- **accelerometer (Vector3)** – Acceleration data from the accelerometer.
- **gyroscope (Vector3)** – Angular acceleration data from the gyroscope.
- **magnetometer(Vector3)** – Magnetic flux density data from the magnetometer.

3.1.4. Device State

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

3.1.5. Battery State

- **battery (float)** – Battery fuel gauge level.
- **chargingInProgress (bool)** – Boolean indicating if the device is plugged in and charging (true) or not (false).
- **chargingComplete (bool)** – Boolean indicating if the device has finished charging (true) or not (false).

3.2. 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 as follows: *bit_offset (byte, bit), length_in_bits*.

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
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

4. Usage

4.1.1. Retrieving input data

The eteeAPI.cs script allows you to call and retrieve device data using Get commands, such as <*GetAllFingersPull()*>, which returns each of the finger's pull data from either the left or right device.

```
using System.Collections;
using UnityEngine;

public class TestScript : MonoBehaviour {
    public eteeAPI api;

    private void Update()
    {
      // Retrieve left and right hand finger pull data
      float[] leftPullData = api.GetAllFingersPull(0);
      float[] rightPullData = api.GetAllFingersPull(1);

      // Print index values in terminal
      Debug.Log("Left Index Pull Pressure: " + leftPullData[1]);
      Debug.Log("Right Index Pull Pressure: " + rightPullData[1]);
    }
}
```

The eteeAPI script also contains **Is** functions that check the status conditions of connected devices. An example is the <IsAnyDeviceConnected()>, which checks if either controller device is connected.

```
using System.Collections;
using UnityEngine;

public class TestScript : MonoBehaviour {
    public eteeAPI api;

    private void Update()
    {
      // Example on how to check connection status of devices
      bool anyControllerConnected = api.IsAnyDeviceConnected();

      bool leftControllerConnected = api.IsLeftDeviceConnected();
      bool rightControllerConnected = api.IsRightDeviceConnected();
    }
}
```

```
/// <summary>
/// Checks if either controller
/// is connected.
/// </summary>
/// <returns>bool</returns>
0 references
public bool IsAnyDeviceConnected()
{
    if (serialRead.IsDeviceConnected(0) || serialRead.IsDeviceConnected(1))
    {
        return true;
    }
    else
    {
        return false;
    }
}
```

4.1.2. Hand-finger visualisation

After importing the etee API, you can drag the eteeAPI prefab into your scene for immediate use.

To visualise the finger curling, you can follow either option:

- **Default**: Bring the LeftHandController / RightHandController prefabs into the scene. This prefab offers a plug-and-play solution for hand visualisation, using a default hand model.
- **Custom**: If you wish to use your own hand model, just attach the HandController.cs script to your model. Make sure to reference the correct Device and API references in the Inspector Panel.

4.2. Creating Connections and Streaming Data

A serial connection between the driver and the eteeDongle is established by calling the **EnableDataStreaming()** function.

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

The **StartThread()** function 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.

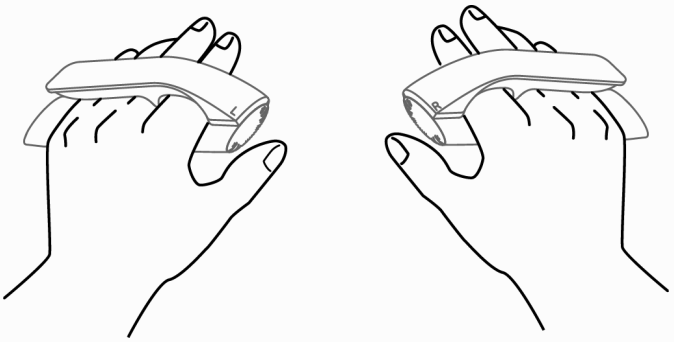
4.3. Resetting Sensor Baselines

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).

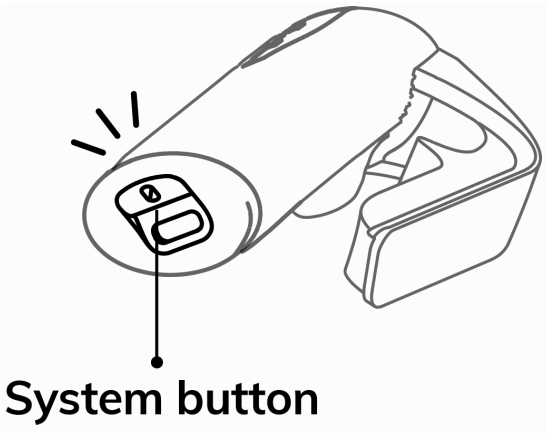
Steps for quick calibration:

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



2. **Trigger quick calibration** by using the physical button.

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.