A USB HID Keyboard, Mouse, Touchscreen emulator with Teensy

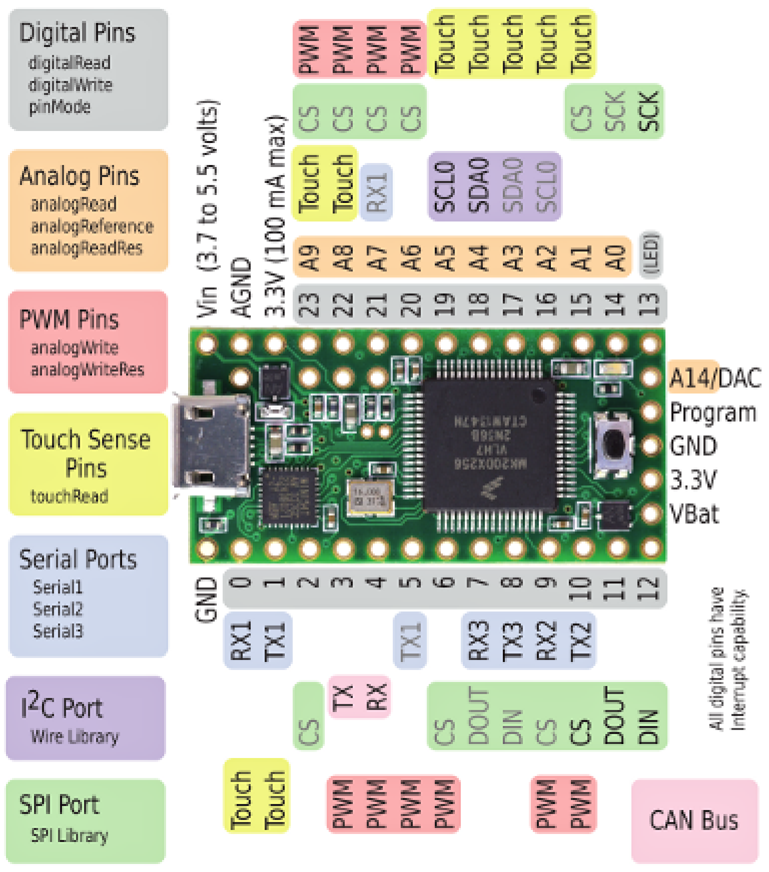
Elmue

13 Nov 2017 Apache

This article describes how to use a Teensy 3.1 board from PJRC.com to simulate keyboard, mouse and touch screen USB HID devices at the same time. This allows to remote-control a computer through one USB cable.

Download sourcecode - 34.6 KB

Teensy 3.1



The Teensy 3.1 is a tiny PCB board (35 x 18 mm) with a lot of hardware functionality. It can be programmed with the same developer environment as an Arduino board. Among the features of the Teensy is the capability of the microprocessor to emulate USB devices.

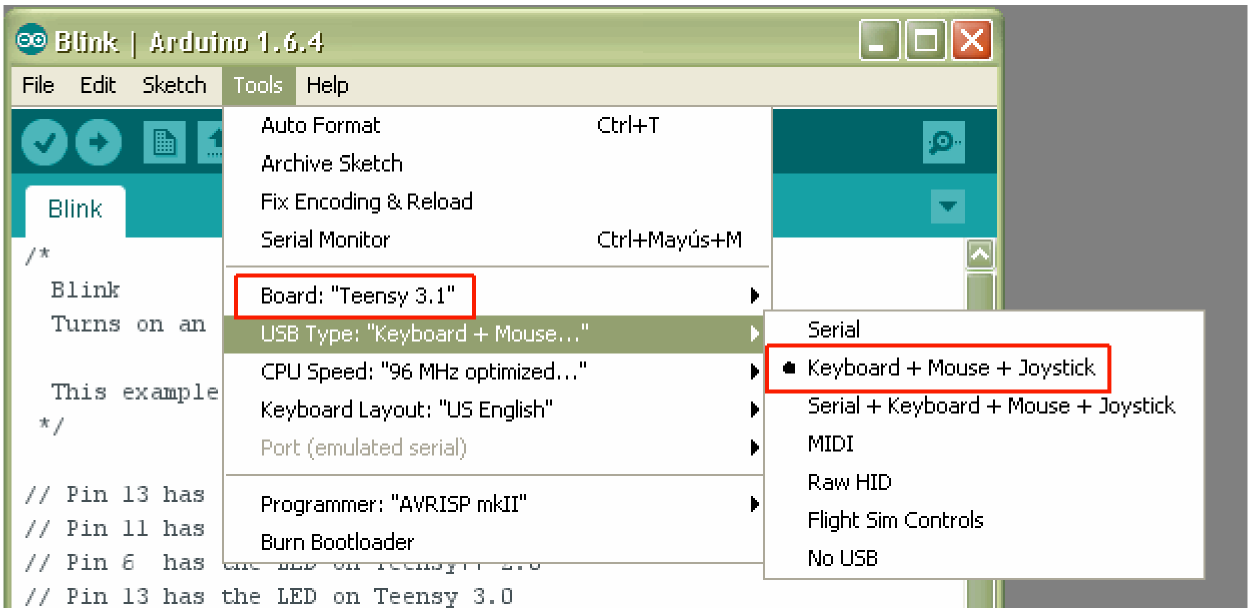
Programming

To program the Teensy's processor you need a micro USB cable and you have to install:

1. The Arduino compiler version 1.67. (Download)
2. The TeensyDuino libraries (Download)
   Direct links to version 1.27: MAC, Linux 32 bit, Linux 64 bit, Windows

The source code has to be written in C and the firmware that you write for the board is named a "*Sketch*".

You have to configure the compiler like in this screenshot:



Board: "Teensy 3.1"
USB Type: "Keyboard + Mouse..."
Keyboard + Mouse + Joystick
CPU Speed: "96 MHz optimized..."
Keyboard Layout: "US English"
Programmer: "AVRISP mkII"

```
/*
  Blink
  Turns on an

  This example
 */

// Pin 13 has
// Pin 11 has
// Pin 6  has
// Pin 13 has the LED on Teensy 3.0
```

You can also use the option "**Serial + Keyboard + Mouse + Joystick**" which will add an USB virtual COM port to the keyboard, mouse and joystick devices.

Problems with the actual version TeensyDuino 1.27

When you have installed and configured the compiler you can already start writing code that emulates keyboard, mouse and joystick. You will find several example projects already installed on your harddisk (see menu "File" -> "Examples").

But there is a severe problem with the original code from TeensyDuino 1.27:

The mouse emulation does not work on Linux.
The cause is a severe design error in the Linux X11 server which is full of bugs and misdesigns.

The X11 server does not accept a USB mouse device that sends absolute coordinates.
But if you want to position the mouse at an exact location on the screen, the mouse positioning with relative coordinates is useless for several reasons:

1. As you don't know the current location of the mouse pointer on the computer that is connected to the Teensy you would have to first move the mouse to the top left corner of the screen and start a relative movement from there, which is ugly.
2. When the Teensy USB device tells the remote computer that the mouse has moved by 100 units it depends on the settings in control panel and on the operating system, how much the mouse really moves on the screen. It may move 53 pixels or 144 pixels!
3. When "Mouse Enhancement" is enabled in control panel it becomes even worse: The distance of movement will depend additionally on the movement speed.
4. Relative movement works on Linux only correctly in steps of 1 due to bugs/misdesigns in X11.
5. Summary: If you want to position the mouse pointer exactly with relative movement: FORGET IT !

For that reason PJRC has implemented absolute mouse positioning in the versions TeensyDuino **1.17** and **1.18**. This works better but then some users complained that they now cannot move the mouse anymore relatively which they needed for their projects.

So in the TeensyDuino versions **1.19** to **1.27** PJRC implemented a USB device that allows relative **and** absolute movement in the same HID device. But Linux does not accept this HID device and the mouse does not move with absolute coordinates anymore.

SUMMARY:
Whatever version of TeensyDuino you use, you will never have a code that satisfies the needs of all users and all operating systems.

My changes

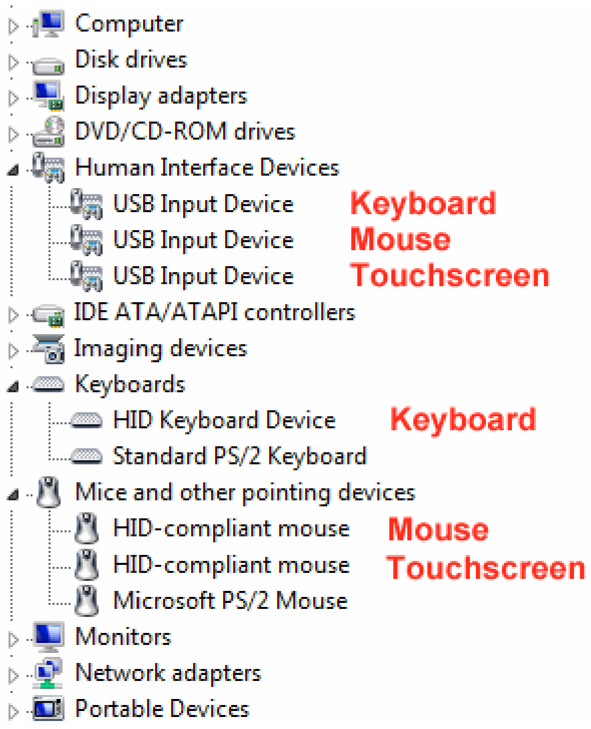
And here comes my project. The code that you can download above in the ZIP file is a modification of the Teensyduino 1.27 code.
Copy the 5 files in the ZIP to the folder ArduinoCompiler/hardware/teensy/avr/cores/teensy3
I made the following changes to the original code:

1. The **mouse device** uses ONLY relative coordinates which will work on all operating systems (just like a "normal" real mouse).
2. I added a new **touch screen device** for absolute pointer movement that also works on all operating systems.
3. I did **not** change the commands of the Mouse class that you already are using in your current sketches, so you simply replace the files in the ZIP file above and recompile your sketch without changes.
4. I completely rewrote the **Mouse class**, cleand up the code and added more functionality.
5. I added lots of **comments** to the code that was sparsely commented before.
6. I have commented out the **joystick** device which I don't need. If you need it, read my comments in **usb_desc.h** how to enable it again.
7. I removed the **SerialEmu** device which I don't need. If you should need it, you have to change only a very few lines in **usb_desc.h**.

I hope that PJRC will integrate my code into future versions of TeensyDuino but currently USB stuff is in a feature freeze.

For the case that PJRC does not implement my code, you find also the original code from 1.27 in the ZIP file. So you can compare what I have changed and you can also compare the orginal 1.27 code with any future version to see what PJRC has changed. (I recommend Araxis Merge to compare text files)

After applying my changes and connecting the programmed Teensy to a Windows computer you will see in control panel 3 new USB devices:



USB device descriptors

To create a new HID device in Teensy I had first to learn a lot about USB descriptors. There is no really well written manual about this complex stuff in internet. If you read the documents from USB.org you will get shocked about the horrible writing quality. You will probably not understand anything.

You find some more user friendly tutorials at Beyondlogic and Eleccelerator but even these stay cryptic for a beginner.

A VERY great help for Windows users is the program USBlyzer which exists as a 33 day trial version with full functionality. This **USB Sniffer** shows all device descriptos of all connected USB devices and the data packets sent through the USB cable(s).

To be sure that what I do is correct I copied the device descriptors of a real **Genius** optical mouse and a real touch screen: **ELO** TouchSystems CarrollTouch 4500U. The keyboard device stays unchanged.

Here are the HID device descriptors that I finally implemented:

Mouse

```
Usage Page (Generic Desktop)
Usage (Mouse)
Collection (Application)
    Usage Page (Button)
    Usage Minimum (Button 1)
    Usage Maximum (Button 3)
    Logical Minimum (0)
    Logical Maximum (1)
    Report Count (3)
    Report Size (1)
    Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
    Report Count (1)
    Report Size (5)
    Input (Cnst,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
    Usage Page (Generic Desktop)
    Usage (Pointer)
    Collection (Physical)
        Usage (X)
        Usage (Y)
        Logical Minimum (-32767)
        Logical Maximum (32767)
        Report Size (16)
        Report Count (2)
        Input (Data,Var,Rel,NWrp,Lin,Pref,NNul,Bit)
    End Collection
    Usage (Wheel)
    Logical Minimum (-127)
    Logical Maximum (127)
    Report Size (8)
    Report Count (1)
    Input (Data,Var,Rel,NWrp,Lin,Pref,NNul,Bit)
End Collection
```

Touchscreen (version 'Pointer')

```
Usage Page (Generic Desktop)
Usage (Pointer)
Collection (Application)
    Usage Page (Generic Desktop)
    Usage (Pointer)
    Collection (Physical)
        Usage Page (Button)
        Usage Minimum (Button 1)
        Usage Maximum (Button 1)
        Logical Minimum (0)
        Logical Maximum (1)
        Physical Minimum (0)
        Physical Maximum (1)
        Unit (None)
        Report Size (1)
        Report Count (1)
        Input (Data,Var,Abs,NWrp,Lin,NPrf,Null,Bit)
        Report Size (1)
        Report Count (7)
        Input (Cnst,Ary,Abs)
        Usage Page (Generic Desktop)
        Usage (X)
        Usage (Y)
        Logical Minimum (0)
        Logical Maximum (10000)
        Physical Minimum (0)
        Physical Maximum (10000)
        Unit (None)
        Report Size (16)
        Report Count (2)
        Input (Data,Var,Abs,NWrp,Lin,NPrf,Null,Bit)
    End Collection
End Collection
```

As you see the **mouse** now allows **relative** movement of 16 bit (-32767 to 32767) rather than 8 bit (-127 to 127) in the original code which was far too few.

The **touch screen** uses coordinates from 0 to 10000 which allows to pass values in **percent** of the screen width/height with a precision of two digits after the decimal point. So to position the mouse at the bottom right corner of the screen the Mouse class sends the values 100.00% for X and Y which corresponds to a value of 10000 beeing sent through the USB cable. To position the mouse in the center of the screen the Mouse class sends 5000, 5000.

Note that it is completely **irrelevant** how many pixels the real monitor resolution is. This works on ANY monitor size because the operating system that receives the touch screen coordinates converts them into screen coordinates in pixels. All touch screens work like this.

I tested my project on Windows XP, 7, 8, 10 and on Linux (Suse, Ubuntu, Knoppix) where it works seamlessly.

First alternate Touchscreen Descriptor

Update february 2016:
Some people reported here on Codeproject that the above touchscreen descriptor does not work on OSX and Android. So I created

an alternate descriptor that you must enable in the file `usb_desc.h` by changing a zero into a one in this line:

Hide Copy Code

```
#define TOUCH_DEVICE   1
```

This will replace the above descriptor with this one:

Touchscreen (version 'Single Touch')

Usage Page (Digitizer)
Usage (Pen)
Collection (Application)
  Usage (Stylus)
  Collection (Physical)
    Usage (Tip Switch)
    Usage (In Range)
    Logical Minimum (0)
    Logical Maximum (1)
    Report Size (1)
    Report Count (2)
    Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
    Report Size (1)
    Report Count (6)
    Input (Cnst,Ary,Abs)
    Usage Page (Generic Desktop)
    Usage (Pointer)
    Collection (Physical)
      Usage (X)
      Usage (Y)
      Logical Minimum (0)
      Logical Maximum (10000)
      Physical Minimum (0)
      Physical Maximum (10000)
      Unit (None)
      Report Size (16)
      Report Count (2)
      Input (Data,Var,Abs,NWrp,Lin,Pref,NNull,Bit)
    End Collection
  End Collection
End Collection

I tested this descriptor successfully on Windows XP, 7, 8, 10 and on Ubuntu and Knoppix.

A Codeproject user reports that this descriptor works also on Android.
Please report results on OSX!

Second alternate Touchscreen Descriptor

You can enable the third touchscreen descriptor in the file `usb_desc.h` with:

Hide Copy Code

```
#define TOUCH_DEVICE 2
```

This descriptor is a multi-touch descriptor, but it uses only one finger.

Touchscreen (version 'Multi Touch')

Usage Page (Digitizer)
Usage (Touch Screen)
Collection (Application)
  Usage (Contact Count Maximum)
  Logical Maximum (1)
  Feature (Data,Var,Abs,NWrp,Lin,Pref,NNul,NVol,Bit)
  Usage (Contact Count)

```
    Report Count (1)
    Report Size (8)
    Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
    Usage (Finger)
    Collection (Logical)
        Usage (Contact Identifier)
        Report Size (8)
        Report Count (1)
        Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
        Usage (Tip Switch)
        Usage (In Range)
        Logical Minimum (0)
        Logical Maximum (1)
        Report Size (1)
        Report Count (2)
        Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
        Report Count (6)
        Input (Cnst,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
        Usage Page (Generic Desktop)
        Usage (X)
        Usage (Y)
        Logical Minimum (0)
        Logical Maximum (10000)
        Physical Minimum (0)
        Physical Maximum (10000)
        Unit (None)
        Report Size (16)
        Report Count (2)
        Input (Data,Var,Abs,NWrp,Lin,Pref,NNul,Bit)
    End Collection
End Collection
```

ATTENTION:
This descriptor **behaves very different** from the previous two descriptors.
When you call `Mouse.moveTo()` the mouse pointer will NOT move on the screen.
(The only way to see the mouse pointer moving is inside the drawing area of MSPAINT.EXE on Windows 8 or 10.)

The mouse pointer may even disappear when moving it (Linux and Windows). The reason is that on a real touch screen you do not need the mouse pointer because you see where you are touching, so the operating system hides the mouse pointer.

When you touch with the simulated finger, the mouse pointer jumps to the new location and executes a click.

This descriptor works immediately on Windows XP, 8, 10 and Ubuntu and Knoppix.
But nothing will happen on **Windows 7** if you do not enable it first. Go to Control Panel -> Pen and Touch:

After the first installation (Windows says: "The device is ready now") it may be necessary to disconnect and reconnect the Teensy once to make it work.

Try which of the three touchscreen descriptors works for you by setting `TOUCH_DEVICE` to 0, 1 or 2.

Programming Mouse/Touchscreen Movement

The commands in your sketches can stay unchanged, but you get extended functionality:

Move the mouse pointer with relative coordinates (-32767 to 32767) via **mouse device**:

Hide Copy Code

```
Mouse.move(x, y);
```

Move the mouse pointer with absolute coordinates in percent (0 to 10000) via **touchscreen device**:

Hide Copy Code

```
Mouse.moveTo(x, y);

Example:
Mouse.moveTo(5000, 5000);
moves the mouse to the center of the screen (50.00%)

Mouse.moveTo(10000, 10000);
moves the mouse to the bottom/right corner of the screen (100.00%)
```

Move the mouse pointer with absolute coordinates in pixels on a screen of 1024 x 768 pixels resolution via **touchscreen device**:

Hide Copy Code

```
Mouse.screenSize(1024, 768);
Mouse.moveTo(x, y);

Example:
Mouse.moveTo(1024/2, 768/2);
moves the mouse to the center of the screen

Mouse.moveTo(1024, 768);
moves the mouse to the bottom/right corner of the screen
```

As you see:
After calling `Mouse.screenSize()` the coordinates automatically change from percent to pixel.
You have to call `Mouse.screenSize()` only once (or never).

Special Case: Apple Macintosh

The previous code of PJRC had a workaround for Apple Macintosh which puts a fix margin of 7.5% around the usable area of the mouse movement resulting in 85% of the screen area. See this posting.

I don't know if a touchscreen on MAC also needs this mouse workaround because I don't have a MAC to test it.
For the case that it should be required, I implemented four optional parameters for `Mouse.screenSize()` that allow to define this margin, but in a more flexible way than the original (very cryptic) PJRC code:

Hide Copy Code

```
Mouse.screenSize(widthPixel, heightPixel, // in pixel
                 marginLeft, marginTop, marginRight, marginBottom); // in percent

Example for MAC:
Mouse.screenSize(1024, 768, 750, 750, 9250, 9250);
```

This line defines a screen size of 1024 x 768 pixels and a margin of 7.50% around the usable area. (100.00% - 7.50% = 92.50%).

This information is based on PJRC.
Can a MAC user please confirm that this workaround is really required?

Clicking

Due to the fact that there is now a mouse device that has 3 mouse buttons (left, middle, right) and a touch screen that has only one "*button*" which is the finger of the user, I added the option to click either with the mouse device or the touch screen device:

Hide Copy Code

```
Mouse.click(button);

Or you can also use
Mouse.press(button);
and
Mouse.release(button);

where button may be MOUSE_LEFT, MOUSE_MIDDLE, MOUSE_RIGHT or TOUCH_FINGER
```

If you want to to hold a button down (for example for a drag and drop operation):

Hide Copy Code

```
Mouse.set_buttons(1, 0, 0, 0); // left mouse button down
Mouse.set_buttons(0, 1, 0, 0); // middle mouse button down
Mouse.set_buttons(0, 0, 1, 0); // right mouse button down
Mouse.set_buttons(0, 0, 0, 1); // finger touching
Mouse.set_buttons(0, 0, 0, 0); // release all buttons and the finger
```

Doing a drag-drop operation with the touchscreen device

Hide Copy Code

```
Mouse.moveTo(startX, startY);  // drag coordinates
delay(10);
Mouse.set_buttons(0, 0, 0, 1); // finger down
delay(10);
Mouse.moveTo(endX, endY);      // drop coordinates
delay(10);
Mouse.set_buttons(0, 0, 0, 0); // finger up
```

This will result in a jumping drag operation rather than a slow drag operation.
To get a more realistic simulation you should write a loop which sends the coordinates in smaller steps.
For example to move from coordinate 0,0 to 100,100:

Hide Copy Code

```
Mouse.moveTo(10,10);
delay(5);
Mouse.moveTo(20,20);
delay(5);
Mouse.moveTo(30,30);
delay(5);
Mouse.moveTo(40,40);
etc..
```

Mouse Scrolling

And finally you can use the mouse wheel (-127 to 127) via **mouse device**:

Hide Copy Code

```
Mouse.scroll(movement);
```

Please read the plenty comments that I have put in **usb_mouse.h**.

More projects with Teensy

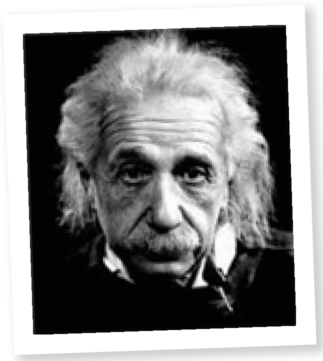
This is my third electronic project with a Teensy published on Codeproject. You may also be interested in:

- DIY electronic RFID Door Lock with Battery Backup
  It uses Desfire RFID cards with DES / AES encryption for an electronic door lock.
- Infrared Remote Control for your Computer
  It uses a Teensy and an IR receiver to control the music player / video player on your computer.

License

Share

About the Author



Elmue

Software Developer (Senior) ElmüSoft
Chile

Watch this Member

Software Engineer since 27 years.

Comments and Discussions

Add a Comment or Question ?   Email Alerts   Search Comments

First Prev Next

Tablet Orientaion
Member 14945166 23-Sep-20 16:00

Re: Tablet Orientaion
Elmue 28-Sep-20 0:43

Re: Tablet Orientaion
Member 14945166 29-Sep-20 5:49

Re: Android Restrictions
Elmue 29-Sep-20 17:17

Re: Android Restrictions
Member 14945166 1-Oct-20 4:56

Fix Android Problem
Elmue 3-Oct-20 18:17

Re: Fix Android Problem
Member 14945166 6-Oct-20 3:35

Sending Touch Inputs
Member 14898273 27-Jul-20 19:13

How to send the data packet
Elmue 28-Jul-20 16:58

Re: How to send the data packet
Member 14898273 30-Jul-20 1:49

Re: Drag and drop
Elmue 1-Aug-20 22:46

Touch Hid Descriptor
Member 14898273 24-Jul-20 23:12

Re: Touch Hid Descriptor
Elmue 27-Jul-20 18:16

Re: Touch Hid Descriptor
Member 14898273 27-Jul-20 19:11

mouse descriptor simulation of cursor
Member 8401297 17-May-20 0:52

Re: mouse descriptor simulation of cursor
Elmue 17-May-20 16:40

Doing a right-click operation on touchscreen device
Member 13984546 25-Jul-19 16:13

Re: Doing a right-click operation on touchscreen device
Elmue 25-Jul-19 19:46

Re: Doing a right-click operation on touchscreen device
Member 13984546 25-Jul-19 20:04

Re: Doing a right-click operation on touchscreen device
Elmue 26-Jul-19 17:57

Re: Doing a right-click operation on touchscreen device
Member 13984546 1-Aug-19 14:30

Re: Doing a right-click operation on touchscreen device
Elmue 2-Aug-19 17:49

Getting usb_keyboard error
Cliffordagius 3-Oct-18 13:50

Re: Getting usb_keyboard error
Elmue 9-Oct-18 16:45

two fingure HID descriptor
Member 13886162 2-Jul-18 17:35

Refresh 1 2 3 4 Next ⊳

General News Suggestion Question Bug Answer Joke Praise Rant Admin

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

Permalink
Advertise
Privacy
Cookies
Terms of Use

Layout: fixed | fluid

Web06 2.8.20201130.1