EETI eGTouch Linux Programming Guide v2.1

TABLE OF CONTENTS

Sec 1: Introduction

EETI provides all kinds of touch solution. EETI eGTouchD is a touch daemon driver for EETI touch controller. Support USB & RS232 interface. And is Available for kernel 3.0 upward. Having below features:

1. **Precise points.**
2. **Report multi points which follow Linux Standard Multitouch-protocol.**
3. **Rightclick, beep sound, constant touch filter and several other functions.**
4. **Available for detecting X-window rotation to do rotating coordinate. (Only for x86 system)**
5. **Support multi devices.**
6. **Provide manually modify driver's behavior.**

This document would assist you to install eGTouchD.

Sec 2: Before install

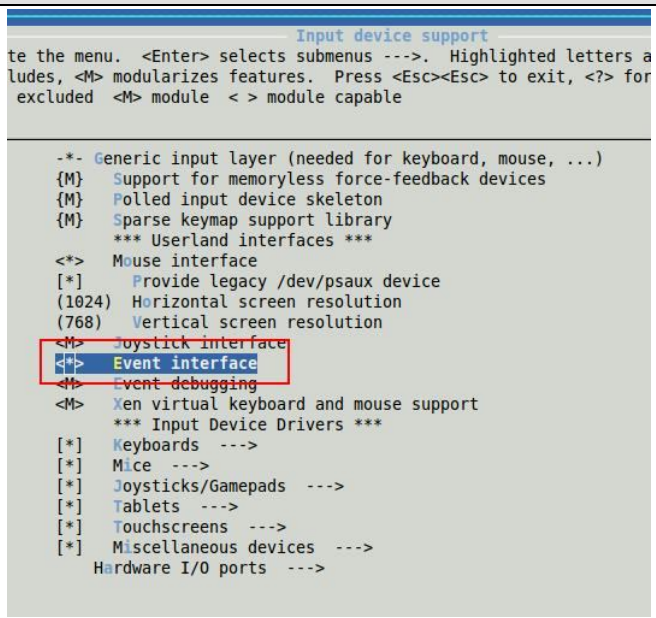
2.1 Patch kernel module

To install driver, we need below kernel modules support:

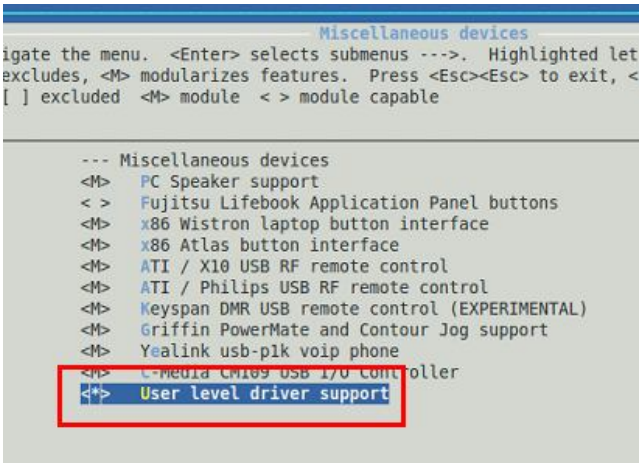
1. **EVDEV**
2. **UINPUT**
3. **HIDRAW (USB Interface)**

If your kernel does not support these features in default, please make sure to enable these functions. Users could check this by "make menuconfig" command or modify Kconfig file. Below is an example of "make menuconfig":

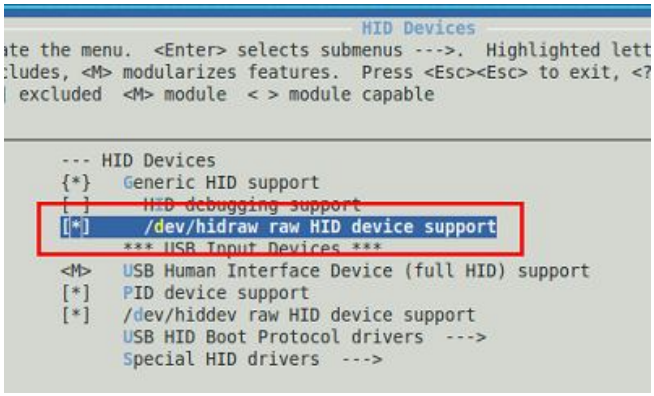
1. [Device Drivers] / [Input device support] / [Event interface]



2. [Device Drivers] / [Input device support] / [Miscellaneous devices] / [User level driver support]



3. [Device Drivers] / [HID Devices] / [/dev/hidraw raw HID device support] **(for USB Interface)**



2.2 Patch kernel source code

Note: If you are using USB interface and your X version is 1.8.7 downwards, you would need to do this part. If not, just skip it.

Append following RED section into your source code.

2.2.a kernel 2.6.33 downwards

1. /SourceCode/drivers/input/**evdev.c**

```
static struct input_device_id evdev_blacklist[] =
{ /* Added by EETI */
        {
        .flags  = INPUT_DEVICE_ID_MATCH_BUS | INPUT_DEVICE_ID_MATCH_VENDOR,
        .bustype = BUS_USB,
        .vendor = 0x0EEF,
        },
        {},     /* Terminating entry */
};

static struct input_handler evdev_handler = {
        .event = evdev_event,
        .connect = evdev_connect,
        .disconnect = evdev_disconnect,
        .fops = &evdev_fops,
        .minor = EVDEV_MINOR_BASE,
        .name = "evdev",
        .id_table = evdev_ids,
        .blacklist = evdev_blacklist, /* Added by EETI */
};
```

2. /SourceCode/drivers/input/**mousedev.c**

```
static struct input_device_id mousedev_blacklist[] =
{       /* Added by EETI */
        {
        .flags = INPUT_DEVICE_ID_MATCH_BUS | INPUT_DEVICE_ID_MATCH_VENDOR,
        .bustype = BUS_USB,
        .vendor = 0x0EEF,
        },
        {
        .flags  = INPUT_DEVICE_ID_MATCH_BUS | INPUT_DEVICE_ID_MATCH_VENDOR,
        .bustype = BUS_VIRTUAL,
        .vendor = 0x0EEF,
        },
        {},     /* Terminating entry */
};

static struct input_handler mousedev_handler = {
        .event = mousedev_event,
        .connect = mousedev_connect,
        .disconnect = mousedev_disconnect,
        .fops = &mousedev_fops,
        .minor = MOUSEDEV_MINOR_BASE,
        .name = "mousedev",
        .id_table = mousedev_ids,
        .blacklist = mousedev_blacklist, /* Added by EETI */
};
```

3. /SourceCode/drivers/input/**joydev.c**

```
static const struct input_device_id joydev_blacklist[] =
{
        {
        .flags = INPUT_DEVICE_ID_MATCH_EVBIT | INPUT_DEVICE_ID_MATCH_KEYBIT,
        .evbit = { BIT_MASK(EV_KEY) },
        .keybit = { [BIT_WORD(BTN_TOUCH)] = BIT_MASK(BTN_TOUCH) },
        },      /* Avoid itouchpads and touchscreens */
        {
        .flags = INPUT_DEVICE_ID_MATCH_EVBIT | INPUT_DEVICE_ID_MATCH_KEYBIT,
        .evbit = { BIT_MASK(EV_KEY) },
        .keybit = { [BIT_WORD(BTN_DIGI)] = BIT_MASK(BTN_DIGI) },
        },      /* Avoid tablets, digitisers and similar devices */
        {
        .flags = INPUT_DEVICE_ID_MATCH_BUS | INPUT_DEVICE_ID_MATCH_VENDOR,
        .bustype = BUS_VIRTUAL,
        .vendor = 0x0EEF,
        },      /* Added by EETI */
        { }     /* Terminating entry */
};

static struct input_handler joydev_handler = {
        .event = joydev_event,
        .connect = joydev_connect,
        .disconnect = joydev_disconnect,
        .fops = &joydev_fops,
        .minor = JOYDEV_MINOR_BASE,
        .name = "joydev",
        .id_table = joydev_ids,
        .blacklist = joydev_blacklist,
};
```

2.2.b Kernel **2.6.34** upwards

1. /SourceCode/drivers/input/**evdev.c**

```
static bool evdev_match(struct input_handler *handler, struct input_dev *dev)
{
        /* Avoid EETI USB touchscreens */
        #define VID_EETI 0x0EEF
        if ((BUS_USB == dev->id.bustype) && (VID_EETI == dev->id.vendor))
            return false;
        return true;
}

static struct input_handler evdev_handler = {
        .event = evdev_event,
        .match = evdev_match,  /* Added by EETI*/
        .connect = evdev_connect,
        .disconnect = evdev_disconnect,
        .fops = &evdev_fops,
        .minor = EVDEV_MINOR_BASE,
        .name = "evdev",
        .id_table = evdev_ids,
};
```

2. /SourceCode/drivers/input/**mousedev.c**

```
static bool mousedev_match(struct input_handler *handler, struct input_dev *dev)
{
        /* Avoid EETI USB touchscreens */
        #define VID_EETI 0x0EEF
        if ((BUS_USB == dev->id.bustype) && (VID_EETI == dev->id.vendor))
            return false;
        /* Avoid EETI virtual devices */
        if ((BUS_VIRTUAL == dev->id.bustype) && (VID_EETI == dev->id.vendor))
            return false;
        return true;
}
static struct input_handler mousedev_handler = {
        .event = mousedev_event,
        .match = mousedev_match,   /* Added by EETI */
```

```
        .connect = mousedev_connect,
        .disconnect = ousedev_disconnect,
        .fops = &mousedev_fops,
        .minor = MOUSEDEV_MINOR_BASE,
        .name = "mousedev",
        .id_table = mousedev_ids,
};
```

3. /SourceCode/drivers/input/**joydev.c**

```
static bool joydev_match(struct input_handler *handler, struct input_dev *dev)
{
        /* Avoid touchpads and touchscreens */
        if (test_bit(EV_KEY, dev->evbit) && test_bit(BTN_TOUCH, dev->keybit))
            return false;
        /* Avoid tablets, digitisers and similar devices */
        if (test_bit(EV_KEY, dev->evbit) && test_bit(BTN_DIGI, dev->keybit))
            return false;
        /* Avoid EETI virtual devices */
        #define VID_EETI 0x0EEF
        if (( BUS_VIRTUAL == dev->id.bustype) && (VID_EETI == dev->id.vendor))
            return false;
        return true;
}

static struct input_handler joydev_handler = {
        .event = joydev_event,
        .match = joydev_match,
        .connect = joydev_connect,
        .disconnect = joydev_disconnect,
        .fops = &joydev_fops,
        .minor = JOYDEV_MINOR_BASE,
        .name = "joydev",
        .id_table = joydev_ids,
};
```

2.3 check device

1.) After patching kernel, build new kernel and reboot for taking effect on changes.

2.) Check kernel functions enable or not

a. UINPUT device node

```
File  Edit  View  Terminal  Help
root@william-desktop:/dev/input# pwd
/dev/input
root@william-desktop:/dev/input# ls uinput -al
crw-r----- 1 root root 10, 223 2010-01-05 15:43 uinput
root@william-desktop:/dev/input#
```

For USB device. After plug in an USB device.

b. HIDRAW device node

```
File  Edit  View  Terminal  Help
root@william-desktop:/dev# pwd
/dev
root@william-desktop:/dev# ls hidraw* -al
crw-rw---- 1 root root 251, 0 2010-01-05 17:02 hidraw0
root@william-desktop:/dev#
```

c. USB touch device handlers

cat /proc/bus/input/devices

```
I: Bus=0003 Vendor=0eef Product=720c Version=0100
N: Name="eGalax Inc. USB TouchController"
P: Phys=usb-0000:00:1d.0-2/input0
S: Sysfs=/devices/pci0000:00/0000:00:1d.0/usb2/2-2/2-2:1.0/input/input7
U: Uniq=
H: Handlers=
B: EV=1b
B: KEY=421 0 30001 0 0 0 0 0 0 0
B: ABS=100 3f
B: MSC=10
```

Sec 3: Install Process

This section describes how to install eGTouch into OS. Before following this, please make sure referring to "**Section 2 Patch Kernel**" to rebuild kernel for supporting necessary features.

Please execute script file **setup.sh** to automatically decompress and install driver. It will help you accomplish below efforts. You could also manually complete those steps.

1. Decompress eGTouch package which contains:
   - a) eGTouchD: a daemon service driver for EETI touch controller.
   - b) eGTouchL.ini: a parameter list could be loaded by driver
   - c) GetEvent.c: a sample code describes how to read EETI input event.
   - d) eGTouchU: an utility tool of eGTouch (Only x86 CPU would contain this)
   - e) eCalib: a calibration tool for embedded Linux system. (Only embedded system would contains this)
2. Place "eGTouchL.ini" into Linux system directory "/etc/eGTouchL.ini" where driver would load it. We can change driver behavior by modifying this file. **The detail descriptions of parameters are described in Section 5.** (You can see brief definitions in eGTouchL.ini)
3. Place eGTouchD & eGTouchU(or eCalib) under /usr/bin. Write in execution /usr/bin/eGTouchD in /etc/rc.local to make eGTouchD executing at system start.
4. After launching eGTouchD, check /proc/bus/input/devices file and we will find two virtual devices called "eGalaxTouch Virtual Device for Multi" and "eGalaxTouch Virtual Device for Single". Information like below figure:

```
I: Bus=0006 Vendor=0eef Product=0020 Version=0001
N: Name="eGalaxTouch Virtual Device for Multi"
P: Phys=
S: Sysfs=/devices/virtual/input/input13
U: Uniq=
H: Handlers=event10
B: PROP=0
B: EV=b
B: KEY=400 0 0 0 0 0
B: ABS=660800001000003
```

```
I: Bus=0006 Vendor=0eef Product=0010 Version=0001
N: Name="eGalaxTouch Virtual Device for Single"
P: Phys=
S: Sysfs=/devices/virtual/input/input14
U: Uniq=
H: Handlers=event11
B: PROP=0
B: EV=b
B: KEY=30000 0 0 0 0
B: ABS=1000003
```

We could check event node which was assigned to the virtual device and read/get input event through this device node, e.g. /dev/input/eventX.

Sec 4: Touch Input Event Sequence

The eGTouchD daemon sends input event through kernel feature UINPUT so that the client program can get these events from /dev/input/eventX.

There are two kinds of different event sequence depending on the value of the parameter **ReportMode** in eGTouchL.ini.

We provide a sample code **GetEvent.c** to show how the event sequence behaves. Please compile the sample code and execute it corresponding to eGTouchD event node (/dev/input/eventX). You would see the event sequence as panel is touched.

1. **As [ReportMode = 4], which means multi-touch.**

Event sequence would follow the Linux kernel multi-touch protocol.

Kernel version	Event Sequence
2.6.35 downward	Protocol Type A ABS_MT_POSITION_X x[0] ABS_MT_POSITION_Y y[0] SYN_MT_REPORT ABS_MT_POSITION_X x[1] ABS_MT_POSITION_Y y[1] SYN_MT_REPORT SYN_REPORT
2.6.36 upward	Protocol Type B ABS_MT_SLOT 0 ABS_MT_TRACKING_ID 0 ABS_MT_POSITION_X x[0] ABS_MT_POSITION_Y y[0] ABS_MT_SLOT 1 ABS_MT_TRACKING_ID 1 ABS_MT_POSITION_X x[1] ABS_MT_POSITION_Y y[1] SYN_REPORT

You can see the detailed rule described in /Documentation/input/**multi-touch-protocol.txt** under Linux kernel source code.

We also implement some extra events for further functions based on this architecture. Please compile and execute **GetEvent.c** to see how the detailed input event sequence behave as multi-touch happens.

2. **As [ReportMode = 1], which means single touch.**

We would report BTN_LEFT and BTN_EXTRA here.

Below is the sequence of input event.

Type = EV_KEY Code = BTN_LEFT Value = left mouse button state of **first point**, 1: pen down / 0: life off.	Type = EV_KEY Code = BTN_EXTRA Value = the touch state of **second point**, 1: pen down / 0: lift off.
Type = EV_ABS Code = ABS_X Value = the X axis position of **first point**. The range is from 0 to 2047.	Type = EV_ABS Code = ABS_RX Value = the X axis position of **second point**. The range is from 0 to 2047
Type = EV_ABS Code = ABS_Y Value = the Y axis position of **first point**. The range is from 0 to 2047.	Type = EV_ABS Code = ABS_RY Value = the Y axis position of **second point**. The range is from 0 to 2047.
Type = EV_SYNC Code = SYN_REPORT Value = 0 A Sync report event, all data will be valid after this event is received.	

Sec 5: eGTouchL.ini Parameter Explanations

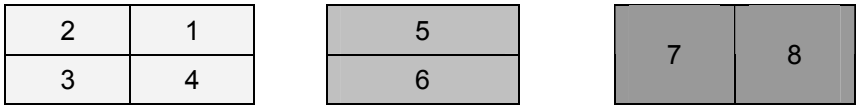
The file **eGTouchL.ini** has a parameter list which would be loaded by driver. Driver's behavior could be changed by these parameters. Please **DON'T** modify the front title as setting up eGTouchL.ini.

If you are using x86 system. There's a tool eGTouchU in this driver package. You could do parameter modification by eGTouchU, which provides a utility interface. Please see detail in document EETI eGTouch Utility Guide for Linux.

5-1 Parameter Table

This table describe the detailed usage of all parameters. There is also a simple description in eGTouchL.ini.

◆ DebugEnableBits	Debug message you want to show.
0	Close all Debug
1	Print initialization debug message **[Default]**
FFFF	Open all Debug
◆ ShowDebugPosition	Position you want to show/store Debug message
0	Print in file located at /tmp **[Default]**
1	Print in terminal
2	Print in above both
◆ Baudrate	Choose the BaudRate
0	Auto detect Baudrate **[Default]**
X	Set Baudrate to X bps. (PCAP: 57600 , Resis: 9600)
◆ SerialPath	RS232 Serial Path
default	Default path /dev/ttySX (X could be equals to 0-10) **[Default]**
/dev/serial/ttyS0	Customized path. Please type in your specific serial path accordinating to the form.
◆ SupportPoints	The amount of points you want to report (This is also confined by Controller)
0	No point
1	Single-touch
>=2	Multi-touch **[Default = 5]**
◆ Direction	Change the X and Y direction
0	Don't make any invert **[Default]**
1	Invert X
2	Invert Y
3	Invert both X and Y
4	Swap X and Y

◆ Orientation	Change the orientation
0 1 2 3	0 degree **[Default]** 90 degree 180 degree 270 degree
◆ EdgeCompensate	Do edge compensate
0 1	Disable **[Default]** Enable
EdgeLeft, EdgeRight EdgeTop, EdgeBottom	Edge compensate value
X	If equals to 100, it means no change. If you set Left=50, you'll see the left-edge points are shrinks inward. And vice versa. **[Min 50 - 150 Max] [Default = 100]**
◆ HoldFilterEnable	Filter out constant touch or not
0 1	Disable **[Default]** Enable
HoldRange	Constant touch valid area
X	±X range of the point which would lead to constant touch **[Min 0 - 50 Max] [Default = 10]**
◆ SplitRectMode	Split the display into Specific Rect. Touch would just show on the specific Rect.
0 1-8 9	No change (Full Display) **[Default]** Driver in-built split Rect  Customized Rect.
CustomRectLeft CustomRectRight CustomRectTop CustomRectBottom	Theses parameters are valid as SplitRectMode=9. You can customize the Rect by these parameters.
0-2047	Four sides of the customized Rect

◆ ReportMode	Decide single or multi event sequence. Also support click on touch and click on release.
1	Single Mouse. Send BTN_LEFT event. Please refer to Sec 4.
2	Send click on Touch. Send click event only as first touch down. Event sequence is ”touch down” and ”touch up”.
3	Send click on Release. Send click event only as touch up. Event sequence is ”touch down” and ”touch up”.
4	Multi Touch Event. Follow multi-touch protocol. Please refer to Sec 4.
◆ DetectRotation (Only for x86 system)	**Enable:** Driver would map its coordinate corresponding to X window rotation. **Please see 5-2 for important note.** **Disable:** If there’s no roation requirement, just disable it.
0	Disable **[Default]**
1	Enable
◆ RightClickEnable	Report mouse Right Click after constant touch for a while
0	Disable Right Click
1	Enable Right Click **[Default]**
RightClickDuration	Constant touch duration to trigger Right Click
X	X milliseconds **[Default = 1500]**
RightClickRange	Valid area of trigger-RightClick constant touch
X	±X range of the point would lead to constant touch for RightClick **[Min 0 - 50 Max] [Default = 10]**
◆ BeepState	Make a beep sound as touch
0	Disable Beep
1	Make a beep sound as ”Touch Down”
2	Make a beep sound as ”Touch Up”
3	Make a beep sound as both two above conditions.
BeepDevice	Choose the beep sound device
0	No device
1	Send beep sound by from system buzzer
2	Send beep sound by from sound card (Only for x86 system)
3	Send beep sound from both devices.
BeepFreq	You can modify buzzer beep frequency here.
X	(Only for buzzer) The buzzer beep frequency. **[Default = 1000]**
BeepLen	You can modify buzzer beep time length here.
X	(Only for buzzer) The buzzer beep time length (ms). **[Default = 200]**

5-2 DetectRotation Note

As DetectRotation is enabled, eGTouch driver have to be executed after X-server is ready.(We use Xlib to do detection). You have to remove the eGTouch execution in rc.local because it would not work out. Please manually put eGTouch execution in the sequence after OS's X server is ready.

We provides **gdm** solution since it's a general startup.

1. Modify the file "**Default**" under **/etc/gdm/Init**
2. Add eGTouch execution **/usr/bin/eGTouchd** at the end of file but before "**exit 0**"
3. Reboot system.

Since the ready time sequence of Xlib is different among diverse startup. We're sorry that we couldn't provide solution correspond to all startup. If there's any further problem as setting up please contact us for technical support.

5-3 Support Rotation and Sound Card Beep for Embedded System

If you are using an embedded system (ex: ARM CPU), and you need support for rotation detection. There's a necessary condition: **Xrandr** lib support, since eGTouch detect rotation event by Xrandr lib.

And son on. If you are using an embedded system (ex: ARM CPU), you need support for sound card beep. There's a necessary condition: **ALSA** lib support, since eGTouch send beep sound by ALSA lib.

If you need this support and your system contains target library, please contact us for a customized driver. Thanks.