EETI eGTouch Programming Guide

On Linux

TABLE OF CONTENTS

Sec 1: Introduction

EETI eGTouch is a touch daemon driver for EETI touch controller. EETI provide different kinds of touch solution. Driver eGTouch supports USB & RS232 interface and is available for kernel 3.0 upward. **In addition to reporting precise points**, eGTouch has extra features:

1. **Trigger RightClick and support filter constant touch.**
2. **Follow Linux Multitouch-protocol which can report more than 2 points.**
3. **Detect X-window rotation to map coordinate.**
4. **Provide manually modify driver's behavior.**

This document would assist you to install eGTouch and describe eGTouch feature in detail.

Sec 2: Kernel Patch Guide (support kernel 3.0)

This section describes how to re-configure and patch Linux kernel to support some kernel features and blacklist inbuilt input driver support for EETI eGTouch daemon driver. It's highly recommended to follow the steps below to re-build Linux kernel before eGTouch driver setup.

2.1 UART Interface

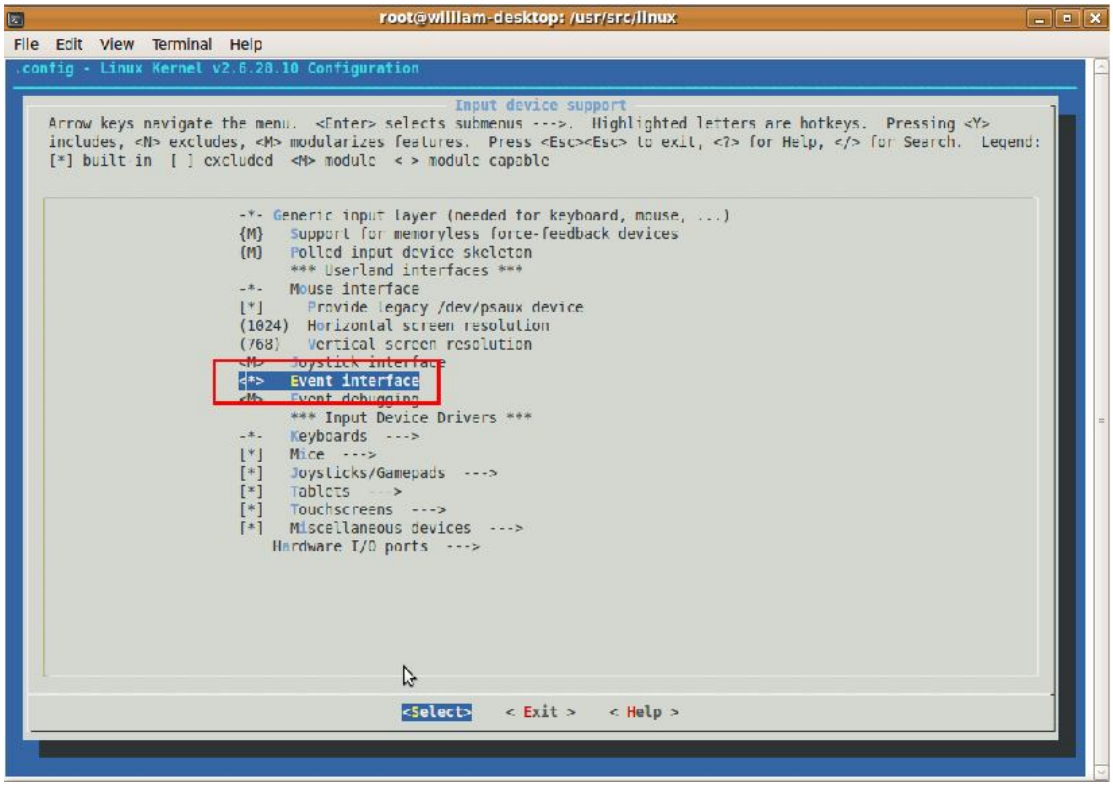
2.1.1 kernel patch

Make sure Linux kernel supports EVDEV and UINPUT two kernel features.

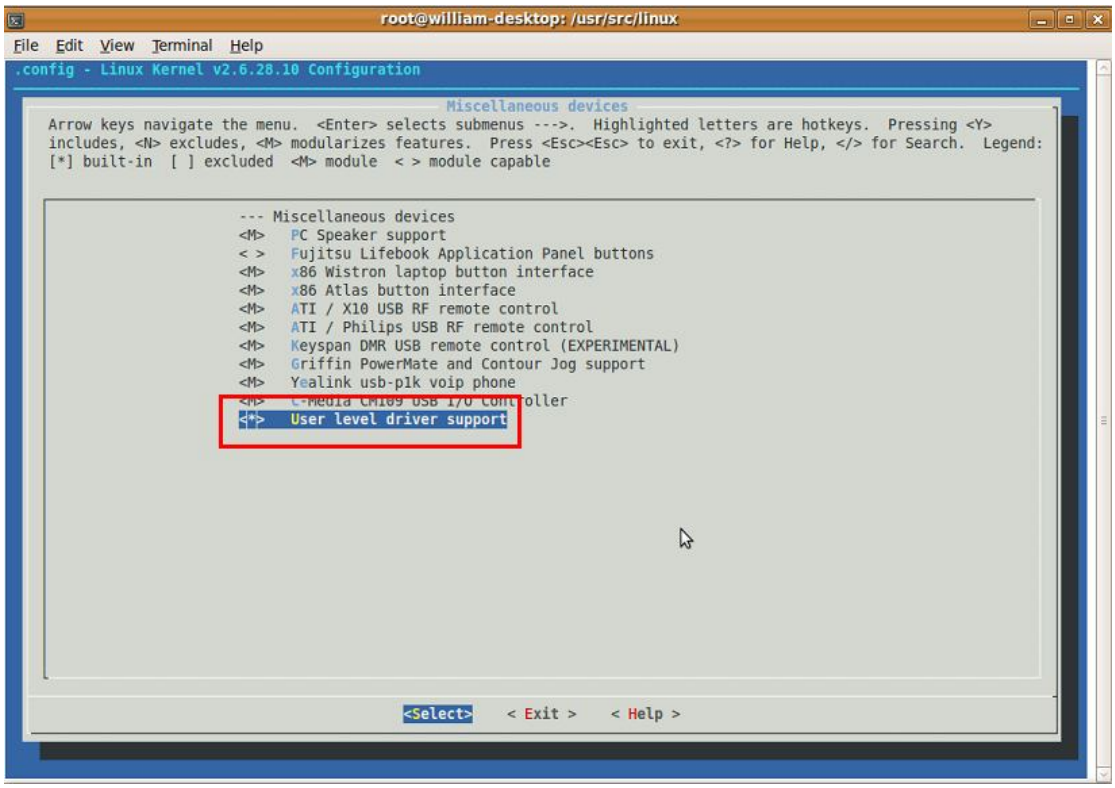
The users could check this by "make menuconfig" command or modify Kconfig file directly.

For example: [menuconfig]

a) Path: [Device Drivers] / [Input device support] / [Event interface]



b) Path: [Device Drivers] / [Input device support] / [Miscellaneous devices] / User level driver support]



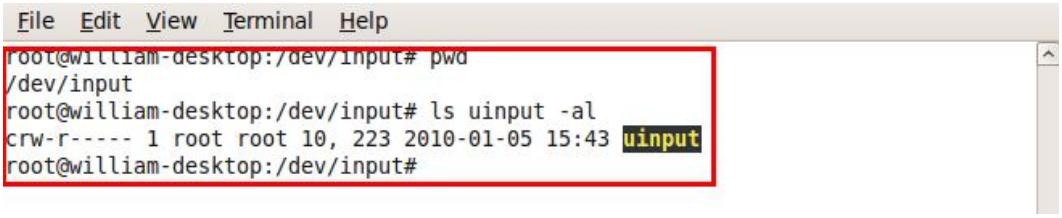
2.1.2 check device

1.) Try to build new kernel and reboot for some changes to take effect.

2.) After boot with new kernel, the users can check whether the following issues the eGTouch driver required is OK or not.

UINPUT device node:

Note: There is a uinput device node created in /dev/ or /dev/input/.



2.2 USB Interface

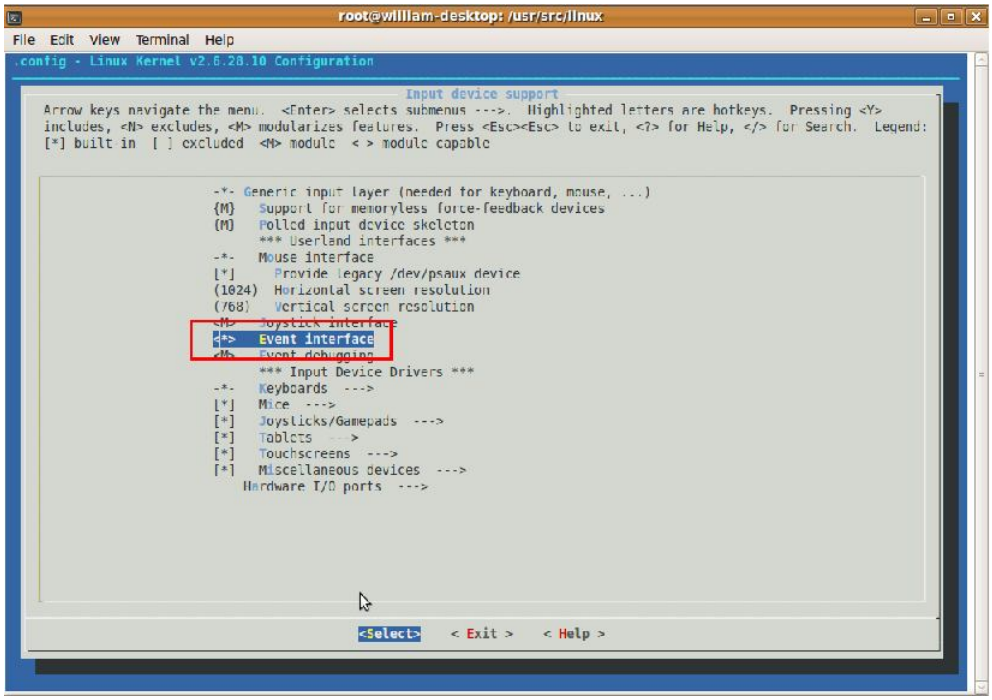
2.2.1 kernel patch

Make sure Linux kernel supports EVDEV, HIDRAW and UINPUT three kernel features.

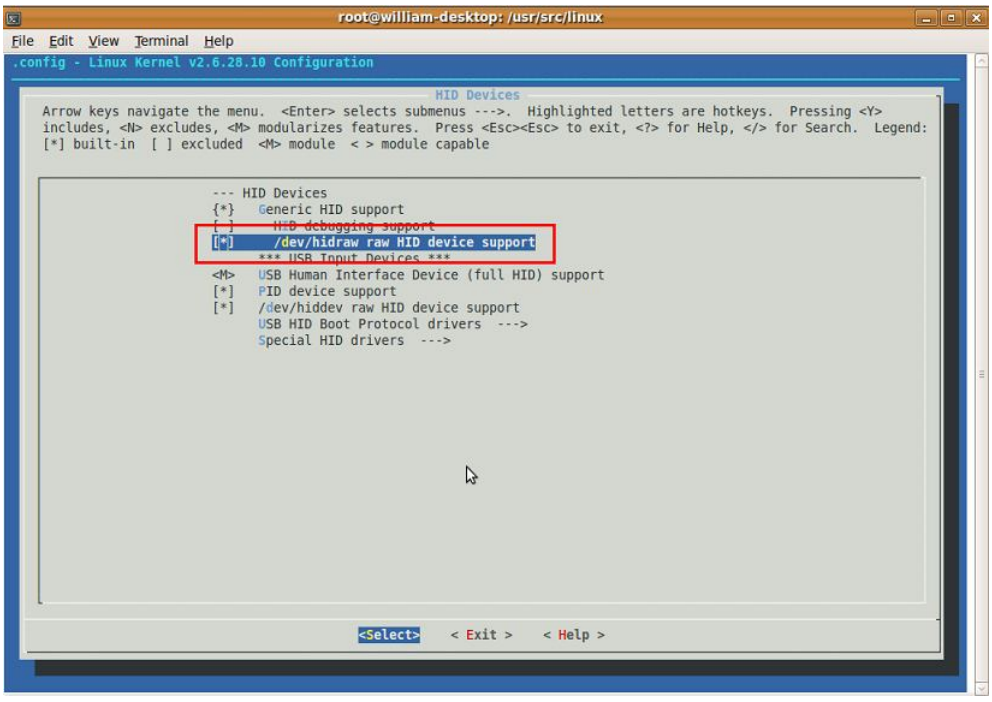
The users could check this by “make menuconfig” command or modify Kconfig file directly.

[Example] for menuconfig

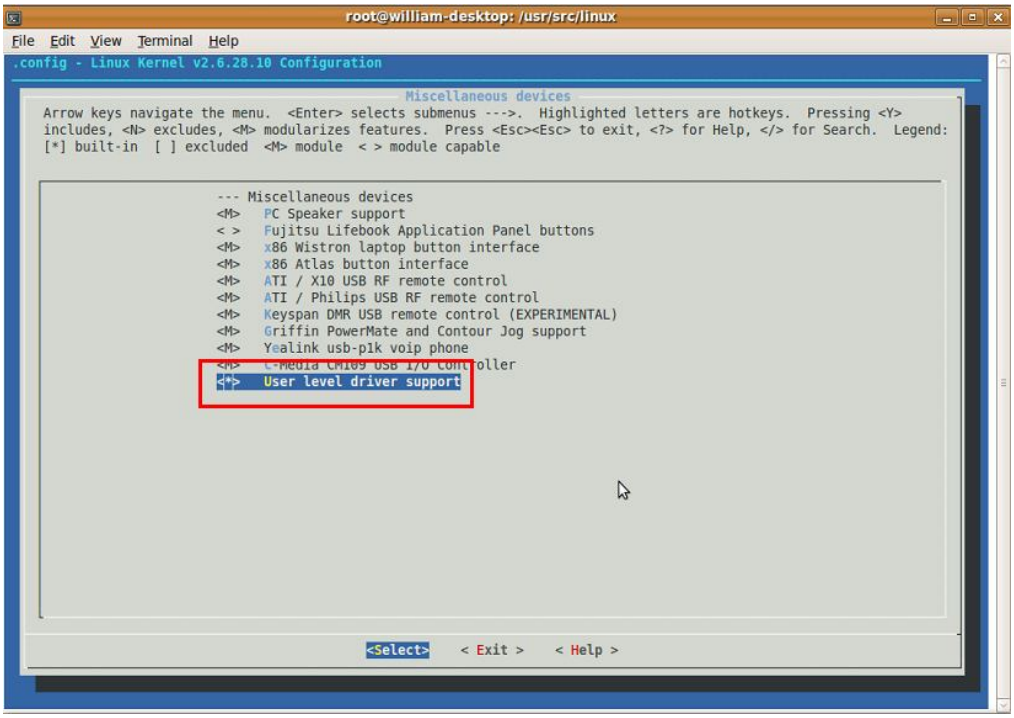
a) Path: [Device Drivers] / [Input device support] / [Event interface]



b) Path: [Device Drivers] / [HID Devices] / [/dev/hidraw raw HID device support]



c) Path: [Device Drivers] / [Input device support] / [Miscellaneous devices] / [User level driver support]



2.2.2 blacklist patch for [kernel 2.6.33 and older]

If X Server version is 1.8.7 above, you can skip this part and move to next step 1.2.4.

(You can check version through command " X –version")

Patch "evdev.c", "mousedev.c" and "joydev.c" to blacklist EETI USB touch device.

Please follow below steps:

a) Patch "evdev.c" as below. Append the following RED code into your source code.

```
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 */
};
```

b) Patch "mousedev.c" as below. Append the RED code below into your source code.

```
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 */
};
```

c) Patch “joydev.c" as below. Append the RED code below into your source code.

```
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,
};
```

2.2.3 blacklist patch for [kernel 2.6.34 and upward]

If X Server version is 1.8.7 above, you can skip this part and move to next step 1.2.4.

(You can check version through command " X –version")

a) Patch "evdev.c" as below. Append the following RED code into your source code.

```
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,
};
```

b) Patch "mousedev.c" as below. Append the following RED code into your source code.

```
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,
};
```

c) Patch "joydev.c" as below. Append the following RED code into your source code.

```
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,
};
```

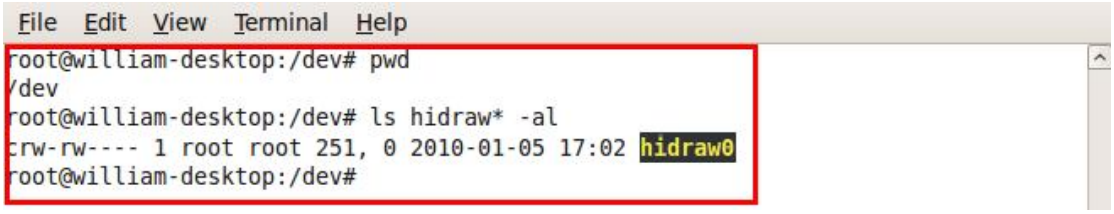
2.2.4 check device

1.) After patching kernel, try to build new kernel and reboot for changes to take effect.

2.) The users can plug an EETI USB touch controller in the system and check whether the following issues eGTouch driver required are OK or not.

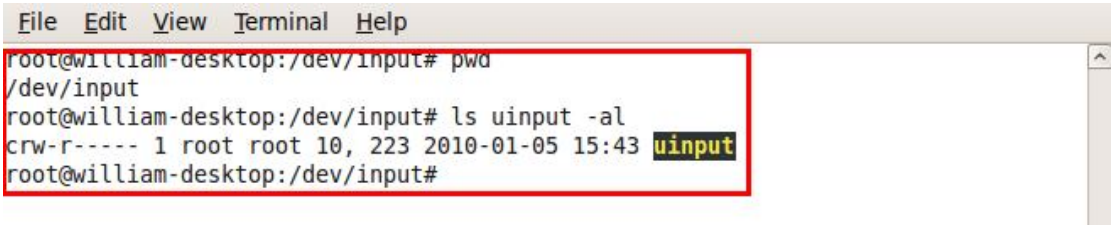
a) HIDRAW device node:

Note: The eGTouch driver will automatically scan and find correct device node related to EETI USB touch controller.



b) UINPUT device node:

Note: There is a uinput device node created in /dev/ or /dev/input/.



c) USB touch device handlers:

Note: After kernel patch, the handler should be empty as below. The information could be checked in /proc/bus/input/devices file.

Sec 3: How To Install

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

3.1 Install steps

Please execute **setup.sh** to automatically install driver package. Through this auto-install, you will complete below effort:

1. Decompress eGTouch package which contains:
   - a) eGTouchd: a daemon service driver for EETI touch controller.
   - b) eGTouchd.ini: a parameter list could be loaded by driver
   - c) eGalaxCalib: calibration tool & drawing test tool
   - d) event.c: a sample code describes how to read EETI input event.
   - e) event_multi.c: a sample code describes how to read EETI multi-touch input event.

2. Place "eGTouchd.ini" into Linux system directory "/etc/eGTouchd.ini" where driver would load it. We can change driver behavior by modifying this file. **The detail descriptions of parameters are described in Section 4.** (You can see brief definitions in eGTouchd.ini)

3. Place eGTouchd & eGalaxCalib under /usr/bin. Make the OS execute eGTouchd driver at starting.

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:

   Note: If the parameter MultiTouchEnable = 0, there would be only "eGalaxTouch Virtual Device for Single"
   (You can see detail description of MultiTouchEnable in Section 4)

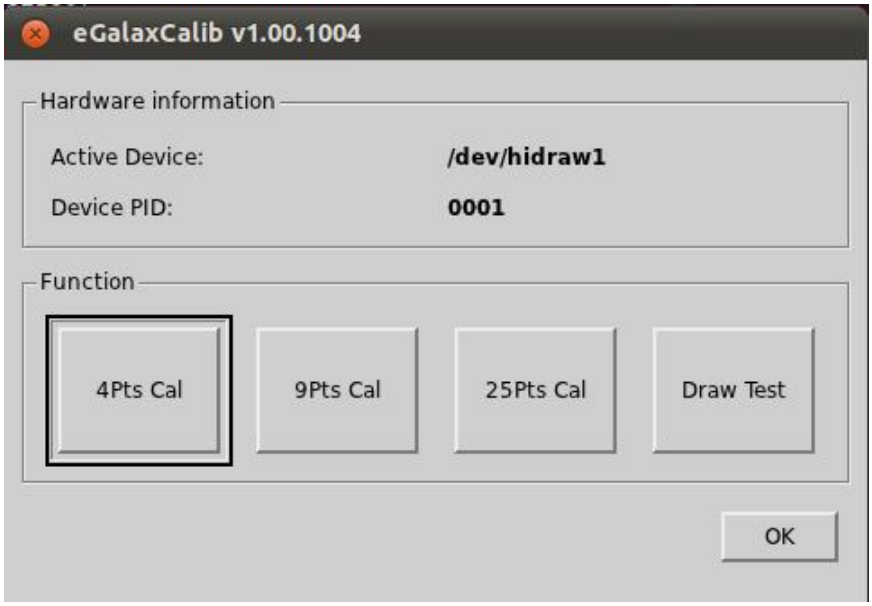
```
I: Bus=0006 Vendor=0eef Product=0020 Version=0001
N: Name="eGalaxTouch Virtual Device for Multi"
P: Phys=
S: Sysfs=/devices/virtual/input/input13
U: Uniq=
H: Handlers=event10
B: PROP=0
B: EV=b
B: KEY=400 0 0 0 0 0
B: ABS=660800001000003

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.

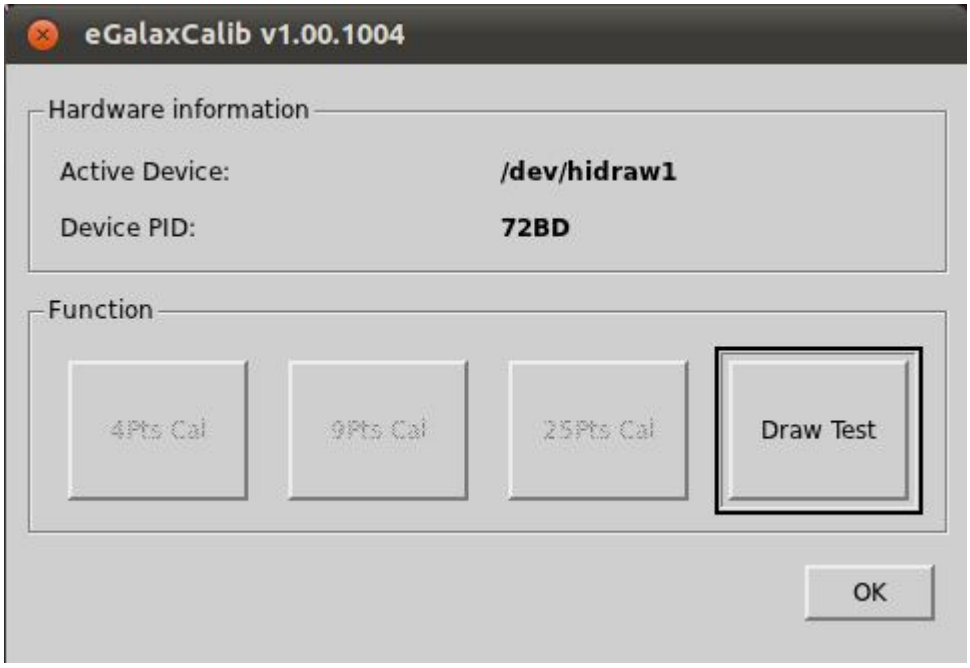
3.2 Tool eGalaxCalib

EETI provides clients with a tool called ”eGalaxCalib” which can do calibration and drawing test. The tool would show a utility as below:

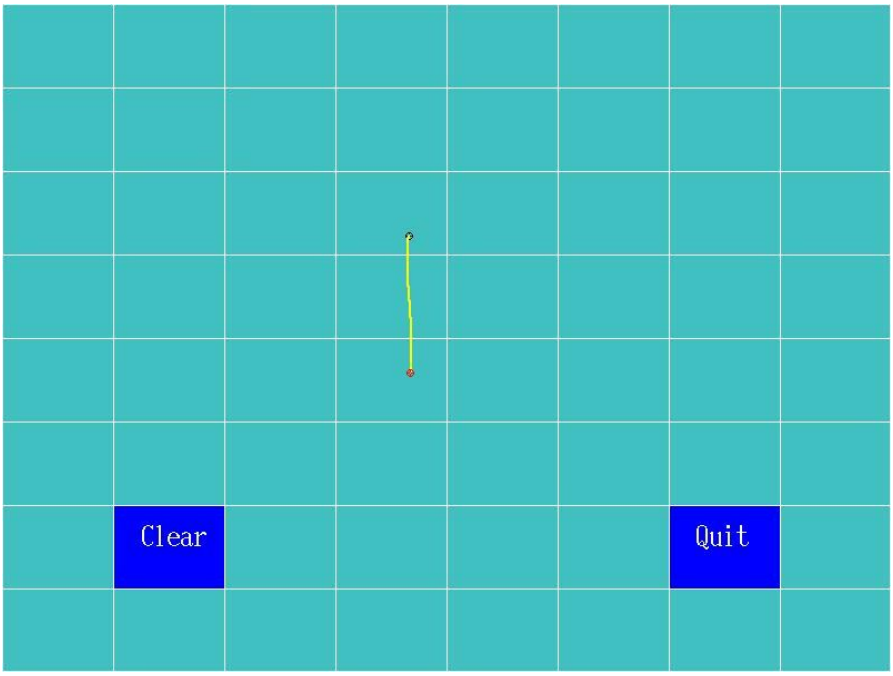
Note: The tool is only available for X window system.



If your touch controller doesn't need do calibration, the calibration button would be in gray and not be executable.



You could execute DrawTest to see drawing outcome.



Sec 4: eGTouchd.ini Parameter Explanations

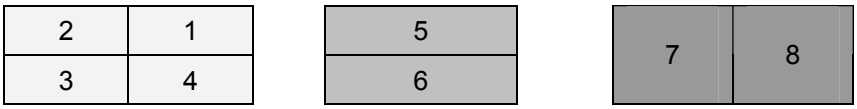
The file **eGTouchd.ini** has a parameter list which would be loaded by Driver eGTouch. Driver's behavior could be changed by setting up these parameters. As setting up eGTouchd.ini, please **DON'T** modify the front title.

4-1 Parameter List

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

1. DebugEnableBits	Debug message you want to show.
0	Close all Debug **[Default]**
255	Open all Debug
2. ShowDebugPosition	**Position you want to show/store Debug message**
0	<Linux> Store in file (under /tmp) <Android> Print in logcat **[Default]**
1	<Linux> Print instantly in terminal <Android> Print in logcat
2	<Linux> Do both <Android> Print in logcat
3. RS232Baudrate	**Choose the BaudRate**
0	57600 bps for EETI serial "Non-Resistive" touch controller **[Default]**
1	9600 bps for EETI serial "Resistive" touch controller
4. MouseMode	**Touch report behavior**
0	Normal mode **[Default]** The touch reports a left button down event when it detects a touch down and a left button up event when it receives a lift off.
1	Desktop mode The driver behaves similar as Normal mode, but the driver will not report mouse button down event immediately after user touches down. The user needs to touch and stay at one point for a few milliseconds, then the driver report mouse touch down event.
5. ReportPoint	**The number of points you want to report** **(This is also confined by Controller)**
0	No point
1	Single-touch
>=2	Multi-touch **[Default = 5]**

6. MultiTouchEnable	**Enable:** (Need kernel above **2.6.36**) Follow Linux kernel Multitouch-protocol. This mode can report more than two points if controller support. **Disable:** report 2 points in max and follow mouse event protocol.
0 1	Disable Enable **[Default]**
7. DetectRotation	**Enable:** Driver would map its coordinate corresponding to X window rotation. **Please see 4-2 for important note.** **Disable:** If there's no roation requirement, just disable it.
0 1	Disable **[Default]** Enable
8. Direction	Change the X and Y direction
0 1 2 3 4	Don't make any invert **[Default]** Invert X Invert Y Invert both X and Y Swap X and Y
9. Orientation	Change the orientation
0 1 2 3	0 degree **[Default]** 90 degree 180 degree 270 degree
10. SerialPath	RS232 Serial Path
default /dev/serial/ttyS0:	Default path /dev/ttySX (X could be equals to 0-10) **[Default]** Customized path. Please type in your specific serial path accordinating to the form.
11. RightClick	Report mouse Right Click after constant touch for a while
0 1	Disable Right Click 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]**

12. 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]**
13. HoldFilterEnable	Filter out constant touch or not
0 1	Disable Enable **[Default]**
HoldRange	Constant touch valid area
X	±X range of the point which would lead to constant touch **[Min 0 - 50 Max] [Default = 10]**
14. 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

4-2 DetectRotation Note

As DetectRotation is enabled, eGTouch driver have to be executed after X-server is ready.(We need Xlib). 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 Xlib-ready sequence 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.

Sec 5: Touch Input Event Sequence

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

There are two kinds of different event sequence depending on whether the parameter **MultiTouchEnable** in eGTouchd.ini is enabled or not.

Both conditions we provide a sample code to help user to show how these events happen in order.

MultiTouchEnable = 0	**event.c**
MultiTouchEnable = 1 **[Default]**	**event_multi.c**

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. struct Input event (defined in /linux/input.h). This is the standard input event structure.

```
struct input_event {
        struct timeval time;
        __u16 type;
        __u16 code;
        __s32 value; };
```

2. **As [MultiTouchEnable = 1] -- event_multi.c**

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

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 event_multi.c to see how the detailed input event sequence behave as multi-touch happens.

3. **As [MultiTouchEnable = 0] -- event.c**

The sequence of input event. We would report BTN_LEFT and BTN_EXTRA here.

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.	