Topics Covered

- Description of Wacom Tablet
- USB HID Device Driver Overview
- Linux Input Event Subsystem Overview
- HID Driver Framework on OpenSolaris
- Wacom Kernel Module and X Input Extension Library on OpenSolaris
Overview of the Wacom Tablet

- Tablet models come in different sizes and features
- Each tablet comes with a pen with replaceable stylus and side switches
- Tablet can send proximity events, absolute pen coordinates, pressure, height, tilt, pen serial number, and various “expresskey” events and slider(s) events
- Tablet contains HID boot protocol which allows pen to work like a mouse
- More information at www.wacom.com
USB HID Device Overview

- Communication between HID devices and a HID driver are in the form of *Device Descriptors* and/or data

- **Device Descriptor**
  - **Configuration Descriptor**
    - **Interface Descriptor**
      - **Endpoint Descriptor**
      - **HID Descriptor**
        - **Report Descriptor**
        - **Physical Descriptor**

- **Descriptors can be viewed using** `mdb(1)` or `prtpicl(1)`

- **See** *Device Class Definition for Human Interface Devices (HID)*
Device Descriptors For Wacom Tablet

# mdb -k

Loading modules: [ unix genunix specfs dtrace mac cpu.generic uppc pcplusmp scsi_vhci zfs sockfs ip hook neti sctp arp usba uhci sd fctl md lofs audiosup fcip fcp random cpc crypto logindmux ptm ufs nsmb sppp ipc ]

> ::prtusb

<table>
<thead>
<tr>
<th>INDEX</th>
<th>DRIVER</th>
<th>INST</th>
<th>NODE</th>
<th>VID.PID</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ehci</td>
<td>0</td>
<td>pci17aa,200b</td>
<td>0000.0000</td>
<td>No Product String</td>
</tr>
<tr>
<td>2</td>
<td>uhci</td>
<td>0</td>
<td>pci17aa,200a</td>
<td>0000.0000</td>
<td>No Product String</td>
</tr>
<tr>
<td>3</td>
<td>uhci</td>
<td>1</td>
<td>pci17aa,200a</td>
<td>0000.0000</td>
<td>No Product String</td>
</tr>
<tr>
<td>4</td>
<td>uhci</td>
<td>2</td>
<td>pci17aa,200a</td>
<td>0000.0000</td>
<td>No Product String</td>
</tr>
<tr>
<td>5</td>
<td>uhci</td>
<td>3</td>
<td>pci17aa,200a</td>
<td>0000.0000</td>
<td>No Product String</td>
</tr>
<tr>
<td>6</td>
<td>scsa2usb</td>
<td>1</td>
<td>storage</td>
<td>1058.0704</td>
<td>External HDD</td>
</tr>
<tr>
<td>7</td>
<td>hid</td>
<td>0</td>
<td>mouse</td>
<td>056a.0065</td>
<td>MTE-450</td>
</tr>
<tr>
<td>8</td>
<td>usb_mid</td>
<td>0</td>
<td>device</td>
<td>0483.2016</td>
<td>Biometric Coprocessor</td>
</tr>
</tbody>
</table>
Device Descriptors For Wacom Tablet (Continued)

> ::prtusb -v -i 7 ← Add “-t” to also show HID Usage Tables

<table>
<thead>
<tr>
<th>INDEX</th>
<th>DRIVER</th>
<th>INST</th>
<th>NODE</th>
<th>VID.PID</th>
<th>PRODUCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>hid</td>
<td>0</td>
<td>mouse</td>
<td>056a.0065</td>
<td>MTE-450</td>
</tr>
</tbody>
</table>

Device Descriptor ← usb_dev_descr_t from uts/common/sys/usb/usbai.h

```c
{
    bLength = 0x12
    bDescriptorType = 0x1
    bcdUSB = 0x200
    BDeviceClass = 0 ← class info in interface descriptor
    bDeviceSubClass = 0
    bDeviceProtocol = 0
    bMaxPacketSize0 = 0x40
    idVendor = 0x56a ← Wacom vendor id
    idProduct = 0x65 ← “Bamboo”
    bcdDevice = 0x108
    iManufacturer = 0x1
    iProduct = 0x2
    iSerialNumber = 0
    bNumConfigurations = 0x1
}
```
Device Descriptors for Wacom Tablet (Continued)

-- Active Config Index 0
Configuration Descriptor
{
    bLength = 0x9
    bDescriptorType = 0x2
    wTotalLength = 0x22
    bNumInterfaces = 0x1
    bConfigurationValue = 0x1
    iConfiguration = 0x0
    bmAttributes = 0x80 ← bus powered
    bMaxPower = 0x16 ← 44mA
}

Interface Descriptor
{
    bLength = 0x9
    bDescriptorType = 0x4
    bInterfaceNumber = 0x0
    bAlternateSetting = 0x0
    bNumEndpoints = 0x1
    bInterfaceClass = 0x3 ← HID Class Device
    bInterfaceSubClass = 0x1 ← Device supports a boot interface
    bInterfaceProtocol = 0x2 ← Boot protocol is mouse
    iInterface = 0x0
}
Device Descriptors For Wacom Tablet (Continued)

HID Descriptor
{
  bLength = 0x9
  bDescriptorType = 0x21 ← Assigned by USB, mouse
  bcdHID = 0x100
  bCountryCode = 0x0 ← Not localized
  bNumDescriptors = 0x1
  bReportDescriptorType = 0x22 ← mouse
  wReportDescriptorLength = 0x92
}

Endpoint Descriptor
{
  bLength = 0x7
  bDescriptorType = 0x5 ← mouse
  bEndpointAddress = 0x81 ← input endpoint number 1
  bmAttributes = 0x3 ← interrupt endpoint
  wMaxPacketSize = 0x9
  bInterval = 0x4
}
Viewing Device Descriptors on Linux

- On Linux, USB device information, including descriptors, is located in /proc/bus/usb/devices
- Information is in ascii (so you can `cat` the file)
- See Documentation/usb/proc_usb_info.txt in the Linux source code
- `lsusb -vvv` also shows descriptors as well as HID Usage Tables
USB HID Device Drivers on Linux

- Drivers for HID devices on Linux can be implemented via:
  - A kernel driver that communicates with a USB host controller driver via the `usb-core` API
    - See Programming Guide for Linux USB Device Drivers
  - A user level driver that communicates with the `hid_input` kernel module
  - A user level driver that communicates with the `hiddev` kernel module
  - `Hid-input` and `hiddev` communicate with the USB host controller driver via `hid-core`
USB HID Device Drivers on Linux (Continued)

- User level drivers communicate with kernel via libusb and/or libhid

- Note that the Wacom implementation on Linux consists of a kernel module that communicates directly with the USB host controller via usb-core
  - User level communication with Wacom is via Linux generic input device (/dev/input/event#)
USB HID Device Drivers on OpenSolaris

- For HID devices, OpenSolaris provides the `hid(7d)` driver and `hidparser` kernel module
  - `hid(7d)` handles all communication with the USB host controller via `usba(7d)` (analogous to `usb-core` on Linux)
  - `hid(7d)` is a STREAMS driver
    - Individual HID devices can use a STREAMS module pushed onto the driver to handle the device
    - There is no documentation for writing such a module
  - The `hidparser` module handles HID descriptors
USB HID Device Drivers on OpenSolaris (Continued)

- OpenSolaris also has support for libusb(3LIB)
  - Uses the ugen(7d) kernel driver to communicate with the USB host controller via usba(7d)
- OpenSolaris currently has no support for libdev or the Linux input device module
- There are currently hid(7d) STREAMS modules to support mouse, keyboard, and audio control devices.
Linux Input Device Handling

- Application opens and reads from an input device (/dev/input/event#, for instance)
- Event Handler is a kernel module that gets input events from the input module
- The input module gets events from registered drivers, and passes them to registered handlers
- The driver handles the device. For USB, the driver communicates with the host controller via usb-core, or via hid-core
- Input events include a time stamp, type of event, code for event type, and a value
  - For instance, a type of event might be a button event, the code indicates which button, and the value would indicate press or release.
/* note that in this example, many details are omitted */
static int foo_probe(struct usb_interface *intf,
        Const struct usb_device_id *id)
{
    struct foo *foo; /* private state data for device */
    foo = kzalloc(sizeof(struct foo), GFP_KERNEL);
    input_dev = input_allocate_device();
    foo->data = usb_buffer_alloc(dev, len, flags, &foo->data_dma);
    foo->irq = usb_alloc_urb(0, flags);
    input_dev->open = foo_open;
    input_dev->close = foo_close;
    /* initialize input_dev capabilities, i.e., */
    /* set input_dev evbits and keybits (buttons, abs vs. rel, etc. */
    /* tell input module about supported and min/max params */
    /* for instance... */
    input_set_abs_params(input_dev, ABS_X, minx, maxx, 0, 0);
    ...  
    endp = intf->cur_altsetting->endpoint[i].desc;
    usb_fill_int_urb(foo->irq, dev,
            usb_rcvintpipe(dev, endp->bEndpointAddress), foo->data, len,
            foo_irq, foo, endp->bEndpointInterval);
    input_register_device(foo->dev);
    /* send/retrieve reports, as needed */
    usb_set_set_report(...);
}
Static void foo_irq(struct urb urb) /* called when data arrives from device (usb-core)*/
{
    struct foo *foo = (struct foo *)urb->context;
    unsigned char *data = foo->data; /* the data from the device */
    struct input_dev *input_dev = foo->inputdev;
    switch(urb->status) {
    case 0:
        /* success, first process data, then send keys, abs/rel, events */
        input_report_abs(input_dev, type, code, value);
        /* and/or input_event(), input_report_rel(), input_report_key() */
        default:
            /* handle error */
    }
}
Linux Input Device Handling – Input Module

- Each input device module maintains bit field arrays of capabilities of the underlying device
  - Device driver fills in bits for corresponding capabilities supported by the device

  - Events
  - Keys
  - Relative Positions
  - Absolute Positions
  - Miscellaneous Events
  - LEDs
  - Sound Effects
  - Force Feedback Events
  - Switches

- Device drivers tell the input module about events that have occurred
  - Input module checks to make sure the device is capable of generating the event
  - Then the input module passes the event to interested event handler(s), or sent to the device (to turn on/off an LED, for instance)

- The input module is meant for generic input device handling, currently only used with usb
Linux Input Device Handling – Event Handler (evdev)

- The `evdev` module is meant for processing of generic events
- Other event handlers exist (mouse, keyboard, joystick), and others can be added
- `evdev` places the event in a client buffer and sends a `SIGIO` to waiting application
- Applications using `evdev` will first open an event device (`/dev/input/event/#` where `#` is between 0 and 31) corresponding to the device for which the application expects events
  - Handler is added for device during `input_register_device()`
  - Applications must search `/dev/input/event/#` devices to find correct corresponding device (open and then get vendor/product id)
Linux Input Device Handling – Application Level

• User level code typically implemented in a library (foo_drv.so)
  1. Applications wishing to use the device link with the library
• For the X windowing system, the library does the following actions:
  1. The ModuleSetupProc function tells X about the new input driver
  2. The PreInit function loads the kernel foo driver and the event handler module (for instance, evdev)
  3. The device_control function, on DEVICE_INIT, opens each /dev/input/event# device until it finds one corresponding to the correct underlying hardware
  4. The read_input function is called whenever packets are ready to be read by the server.
     i. For each packet read, read_input gathers the packets until it has enough information to send event(s) associated with the packet(s)
     ii. Once all packets have been read, the library calls xf86PostxxxEvent() to dispatch button press/release, motion, keystrokes, etc. events to the X server.
• A description of the above functions can be found at http://www.x.org/wiki/Development/Documentation/XorgInputHOWTO
HID Framework on OpenSolaris (Example)

Application

streamhead

Consms STREAMS Multiplexor

usbms

Hid Driver

usba

Host Controller Driver
ehci/ohci/uhci

streamhead

Conskbd STREAMS Multiplexor

usbkbm

Hid Driver
Wacom Tablet on OpenSolaris

• 3 versions
  • Modified usbms module
  • Implement input device handling in kernel module
  • Re-Implement Xinput library module
Wacom Driver as Modified usbms

- Usbms modified to support both mouse and tablet
- All tablet events sent as mouse events
- Version has been in production for 2 years
Problems with First Solution

- The first iteration was implemented because I could not find a way to “pop” the usbms module.
  - Plumbing of `usbms` module done by `consconfig_dacf` kernel module based on `dacf.conf(4)`
    - Boot protocol identifies tablet as a mouse
- No tablet key support (pen, erasor, and side switches work)
- On SPARC, `hwc` module caused problem
- All handling of tablet specific data done by modified `usbms` module
Wacom Module Implementing Linux input Events

- Problem of mouse boot protocol goes away by `open(2)` of the underlying device (`usbms module is popped`)
- `wacom` STREAMS module pushed onto `hid` by modified Linux `wacom_drv.so` library
  - Otherwise, `wacom_drv.so` needs no modification
  - `wacom` module converts raw input from tablet into `input_event` structures expected by library module
  - All features of tablet now work
Problems with Second Solution

- The Linux solution does much of the processing twice, once in the kernel module, and again in the library
- Licensing
  - (But we won't talk about this...)
Wacom on OpenSolaris – Yet Another Solution

- **wacom** kernel module puts the tablet into “pen” mode, and sends raw tablet data to consumers.
- The X input library, *wacom_drv.so*, accepts raw data, converts into X events, and sends the events.
- Currently, solution is using some Linux library code.
  - So, not yet released for OpenSolaris.
Wacom Kernel Module – Sending a Report

wacom_get_vid_pid(wacom_state_t_t wacomp) /* called from module open */
{
    struct iocblk mctlmsg;
    mblk_t *mctl_ptr;
    dev_info_t *devinfo;

    queue_t *q = wacomp->wacom_rq_ptr;

    mctlmsg.ioc_cmd = HID_GET_VID_PID;
    mctlmsg.ioc_count = 0;

    mctl_ptr = usba_mk_mctl(mctlmsg, NULL, 0);

    putnext(wacomp->wacom_wq_ptr, mctl_ptr);
    wacomp->wacom_flags |= WACOM_QWAIT;
    while (wacomp->wacom_flags & WACOM_QWAIT) {
        if (qwait_sig(q) == 0) {
            wacomp->wacom_flags = 0;
            return (EINTR);
        }
    }
}

Return(0);
Wacom Kernel Module – Reading a Report

- The `hid` module acts on `M_CTL` messages and sends another `M_CTL` message upstream.
- The `wacom` module, when it receives the answering `M_CTL` message, takes appropriate action (for instance, waking up code in the open function), and discards the message.
- All tablet data is received as `M_DATA` messages, which are passed upstream with no processing.
Acknowledgements

- Philip Brown's Wacom driver has been helpful, see http://www.bolthole.com/solaris/drivers/usb-wacom.html
- Strony Zhang at Sun Microsystems
- Various people at Wacom
- James McPherson
- The Queensland Brain Institute
- The Source