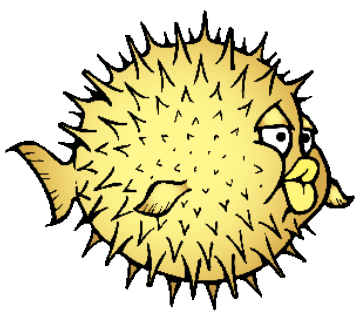
Input devices on OpenBSD

for console, X11 and Wayland

Matthieu Herrb



EuroBSDCon, Brussels, September 12, 2026

License

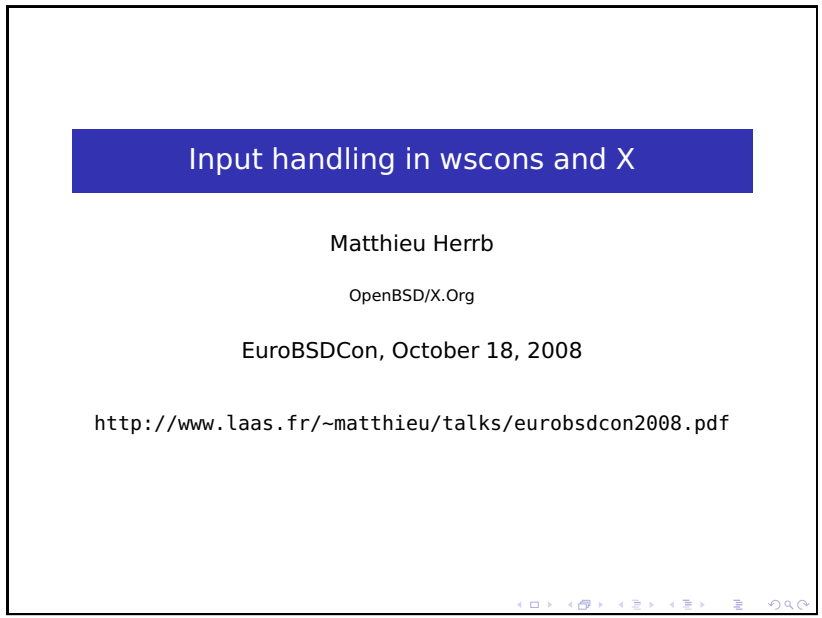
Contents

1. Introduction
2. Improvements in the last 10 years
3. X windows
4. libinput and Wayland
5. wscons evolution
6. Conclusion

Introduction

Old subject...

Already did a talk about it in 2008:



https://homepages.laas.fr/matthieu/talks/eurobsdcon2008.pdf

But still moving (slowly)

Historical Unix: the tty

Serial terminals: text only keyboard and printer or screen

Input : serial (RS232) protocol, line oriented (*cooked* mode: ed,...)

Later: with faster lines : *raw* mode: vi, emacs,....

Graphical screen + keyboard + mouse

Notion of events : key press, release, pointer motion, button clicks,...

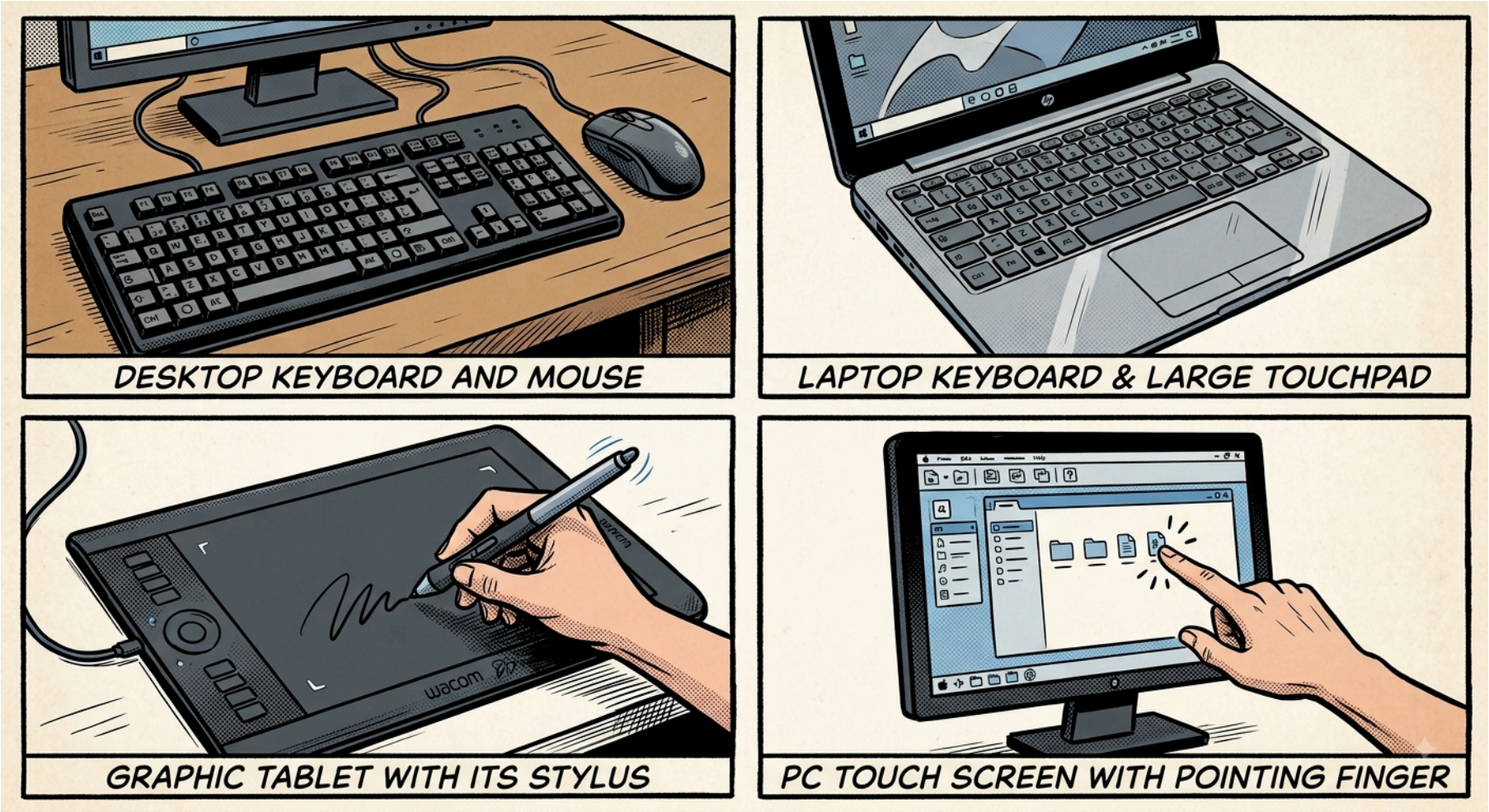
Terminal emulation to keep the old tty-based applications running

Suntools, NeWS, Iris GL,... and later X10 followed by X11

Proprietary (ad-hoc) low level interfaces with the kernel. (still serial-like interfaces)

$\longrightarrow$ pcvt(4) initially for the x86 PC console driver on NetBSD.

Various input devices



- USB : hot-plugabble
- Based on the HID (Human Interface Device) specifications
- Also exist with other attachments: i2c, ...
- Pointing devices have evolved a lot:
  - touchpads,
  - touch screens,
  - multi-touch gestures,
  - fancy scrolling,
  - ...

The wscons driver

Driver for PC -style keyboard + mouse + screen,

Text based terminal emulation using BIOS text mode,
Graphical emulation with bit-mapped graphics otherwise.

Developed for NetBSD before the fork, extended to other architectures and devices

- keyboard: `wskbd`
- mouse: `wsmouse`
- screen: `wsdisplay`
- wsmux: supports for several keyboards or mices
- `/usr/include/dev/wsconsio.h` describes the API / ABI.
- Not much documentation otherwise
- wsconctl(8) handles configuration.

The multiplexer: wsmux

wsmux(4) is the keyboard/mouse multiplexer for wscons

- all wskkbd(4) devices attach to `/dev/wskbd`
- all wsmouse(4) devices attach to `/dev/wsmouse`
- applications only need to open those 2 devices
- the "console" device (`/dev/ttyC0`) also gets wskbd events and handles wsdisplay(4) ioctls for compatibility

When new devices attach (USB) they automatically attach to the mux.

While the general principle is nice and sound there are some issues:

- all keyboards share the same key map
- hard to handle very different pointer devices
  - touch screens needs calibration data
  - touchpad tapping configuration

Keyboards: higher levels

Wscons (and evdev) provide low-level keycodes to the applications.
The xkeyboard-config(7) "standard" allows to translate this to higher levels :

- key map / keysyms (QWERTY, AZERTY, function keys, etc...)
- callbacks (switch VT, key map, terminate, back-light or volume control,...)
- accessibility features (sticky keys,...)

Handled by various pieces of code:

- the XKB extension in the Xorg server
- libxbcommon in Wayland and others

wscons and evdev don't use the same keycodes hence a different set of rules (and bugs)

Improvements in the last 10 years

Advanced touchpad support

Support for multi-touch input to wsmouse

- wstpad driver (bru@ + mpi@ + jcs@)
- tap
- multi-touch
- gestures

Configuration though wsconsctl(8) and wsconsctl.conf(5)

- more protocol decoders (mglocker@, jcs@)
- multi-touch support : hidmt (bru@, jcs@)
- more tablet support in uwacom(4) (Vladimir Meshcheriakov/espie@)

Various

- i2c attachments and drivers : ihidev(4) (jcs@, mglocker@, kettenis@...)
- unlock input (mvs@)
- man bugs and reliability fixes (miod@ and others)
- new keys for Apple Silicium Macs (tobhe@)
- icc/ucc keyboard support (anton@,...)
- suspend/resume fixes and enhancements (deraadt@, kettenis@, mlarkin@,...)

X windows

Current Xorg support state

A bit complex because of legacy code and tech debt...

- kbd(4) Xorg driver reads from `/dev/ttyC4`
  - sets the layout from wscons driver
  - uses `WSDISPLAY_COMPAT_RAWKBD` kernel option to read PC-AT style keycodes

(with extended 2 bytes sequences)

- ws(4) xorg driver for pointer devices.
- detaches devices from the mux

  X handles devices types (touchpad, mouse, touchscreen) differently
- auto-configured at server config time, but no hot-plug
  - uses privileged child to access `/dev/` entries as root

Possible enhancements

General cleanup of the console handling code:

- keep only wscons(4) support
- use `/dev/wskbd` for keyboard input
- get rid of compat layer (WSDISPLAY_COMPAT_RAWKBD) for X.Org drivers
- and switch to evdev compatible keycodes

Hot-plug support:

- add an `WSCONS_EVENT_WSMUX_ATTACH_DEVICE` event to notify the X server of new mux members
- use extended events (see below) to not detach devices from the mux

libinput and Wayland

The core of the Linux input stack

Created by Kristian Høgsberg and Peter Hutterer + numbers of contributors

- gets evdev events from the kernel
- processes them (gestures, multi-touch, acceleration,...)
- handles configuration
- used on Linux by both X.Org and Wayland

Peter's blog : https://who-t.blogspot.com/

port / adaptation of libinput to OpenBSD

- started by mpi, updated by rsadowski@ and volker@
- minimal support for pointers and keyboards
- many stub functions
- used by Wayland ports

On Linux :

- udev handles devices (loads driver into kernel, enumerates them, sends events,...)
- input drivers convert data from hw to evdev, send them to `/dev/input/*`
- libinput makes those events + state available through its API

On OpenBSD (and NetBSD):

- mux(4) handles device attachments and enumeration
- kernel drivers do some processing (acceleration, gestures, multi-touch, key maps)
- `/dev/wskbd` and `/dev/wsmouse` are the only sources of events for applications
- touchscreens and tablets not yet handled

- Compositors based on wlroots ported in 2023
- they use libinput for input
- it would be possible to write a compositor using native wscons API, but applications sill often need libinput.
- current issues:
  - device ownership (need `chown <myuser> /dev/ws*`)
  - mix of different keyboard types (ukbd vs pckbd, aka USB vs PS2)
  - no touchscreen support

Wayland: possible solutions

- unify keycodes for libinput
- handle device access in the display manager (like xendom's `{Take,Give}Console`)
- implement touchscreen support in libinput

wscons evolution

Enhancing events in wscons

disclaimer: none of what follows is currently commited / ok'd in OpenBSD

- wscons has only one fd : the `/dev/wsmouse` mux
- libinput wants to see separate devices
- Solution 1: ~~detach the devices from the mux~~ (not BSDish)
- Solution 2: extend the wsevent struct:
  - provide information about the originating device
  - handle the multiplexing in libinput

wscons extended events

- `struct wscons_event_ex`:

```
struct wscons_event_ex {
    struct timespec time;
    u_int           type;
    int             value;
    u_int           device; /* New */
};
```

  `device` holds the minor device number of the mux member that provided the event
- `WSMUXIO_GTYPE` ioctl(3) : to query the type of a mux member
- `WSKBDIO_SETVERSION` and `WSMOUSEIO_SETVERSION` ioctls to select extended events
- Basic support implemented in libinput-openbsd for Wayland.

wsmux hot-plug events

Need a way to inform Xorg or libinput about added/removed devices

- hotplug(8) is not suited for this
- we don't want to copy / import udev from Linux
- even though robert@ implemented config(8) generation numbers
- so ?

`WSCONS_EVENT_WSMUX_DEVICE` event with 2 values:

- `WSMUX_ATTACH_DEVICE`
- `WSMUX_DETACH_DEVICE`

Tells the application (Xorg or libinput) to re-probe the mux

Support for this event implemented in Xorg and libinput.

Where is the code ?

A version of the `wscons_event_ex` code was sent to wscons developers.

Latest version available here:

https://got.xenocara.org/?action=commits&headref=wscons-event-ex&path=src

The libinput code is on the libopeninput repository in the `openbsd-kbd-ex-mux` branch :

https://github.com/mherrb/libopeninput/commits/openbsd-kbd-ex-mux/

Hope to be merged soon (but probably not before 8.0)

Still missing

Mostly for touch screens / tablets....

- Identifier strings from devices (from USB Vendor / Device IDs or similar)
  - useful for tablet tools ("eraser", "pencil",...)
- Outputs in wsdisplay ("HDMI-1", "eDP-2",...) – hidden by DRM/KMS for now
- Links from wsdisplay outputs to attached input devices
- Configuration mechanism

And some proper use session mechanism to replace fbtab(5) / login_fbtab(3)

(OpenBSD now implements `XDG_RUNTIME_DIR` for user sessions, but it is not enough)

Conclusion

Conclusion

- input is not as simple as it looks
- wscons needs some evolutions
- proposals made

"*A mouse is a device to select which xterm to type into*"
– Dariusz Swiderski

Questions ?

Appendix

Gemini generated images license

You are free to use, modify, and distribute the images generated here for both **personal and commercial purposes**.

Here is how the usage works in practice:

1 Copyright Status

- **No Copyright Claimed:** Google does not claim copyright ownership over the outputs generated by Gemini.
- **Public Domain / Non-Copyrightable:** Under current copyright laws in many jurisdictions (including the US and EU), purely AI-generated artwork generally does not qualify for traditional copyright protection. This means you do not need to pay royalties or request permission to re-use it.

2 Applicable Terms of Service

Your use of these images is subject to [Google's Generative AI Additional Terms of Service](https://policies.google.com/terms/generative-ai) and [Prohibited Use Policy](https://policies.google.com/terms/generative-ai/use-policy).

Key guidelines to keep in mind:

- **Allowed:** Personal projects, educational materials, websites, social media, commercial publications, or design work.
- **Prohibited:** You cannot use the image for illegal activities, generating misleading/harmful content, or violating trademark/copyright rights of existing third-party brands shown or implied (e.g., brand logos appearing in an image).
  - **Transparency:** You should not misrepresent AI-generated content as human-created artwork where transparency or attribution is expected.