

Input Devices & Proton/Wine Gaming

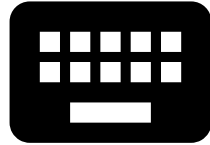
by Arek “ivyl” Hiler

Proton Janitor @  **CODE**
WEAVERS
SOFTWARE LIBERATORS

 @ivyl@treehouse.systems

 ivyl @ libera & oftc

Keyboards & Layout



- The layout detection code has its roots in **1999**.
- **XKB (1996)** was fairly young back then and not supported by all the relevant X11 servers.
- **XKeycodeToKeysym() / XLookupString()**.
- Now using **XkbKeycodeToKeysym() / XkbTranslateKeySym()**.

```

static const WORD main_key_scan_qwerty[MAIN_LEN] =
{
    0x29,0x02,0x03,0x04,0x05,0x06,0x07,0x08,0x09,0x0A,0x0B,0x0C,0x0D,
    0x10,0x11,0x12,0x13,0x14,0x15,0x16,0x17,0x18,0x19,0x1A,0x1B,
    0x1E,0x1F,0x20,0x21,0x22,0x23,0x24,0x25,0x26,0x27,0x28,0x2B,
    0x2C,0x2D,0x2E,0x2F,0x30,0x31,0x32,0x33,0x34,0x35,
    0x56 /* the 102nd key (actually to the right of l-shift) */
};

static const WORD main_key_vkey_qwerty[MAIN_LEN] =
{
    /* NOTE: this layout must concur with the scan codes layout above */
    VK_OEM_3, '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', VK_OEM_MINUS, VK_OEM_PLUS,
    'Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', VK_OEM_4, VK_OEM_6,
    'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', VK_OEM_1, VK_OEM_7, VK_OEM_5,
    'Z', 'X', 'C', 'V', 'B', 'N', 'M', VK_OEM_COMMA, VK_OEM_PERIOD, VK_OEM_2,
    VK_OEM_102 /* the 102nd key (actually to the right of l-shift) */
};

static const char main_key_US[MAIN_LEN][4] =
{
    "`~", "1!", "2@", "3#", "4$", "5%", "6^", "7&", "8*", "9(", "0)", "-_", "=+",
    "qQ", "wW", "eE", "rR", "tT", "yY", "uU", "iI", "oO", "pP", "[{", "]}",
    "aA", "sS", "dD", "fF", "gG", "hH", "jJ", "kK", "lL", ":", ";", "\\", "|",
    "zZ", "xX", "cC", "vV", "bB", "nN", "mM", ",", "<", ">", "/?"
};

{0x0409, "United States keyboard layout", &main_key_US, &main_key_scan_qwerty,
&main_key_vkey_qwerty}, /* 62 of these */

```

winewayland.drv

- More sensible and **less verbose XKB** code.
- Can mature and prove itself as a **mostly clean-slate**.
- Eventually **yoink and twist** for the benefit of winex11.drv *questionmark?*

Mice & Touchscreens



Mice & Touchscreens


XInput2 is *fairly* straightforward

Non-Keyboard & Non-Mice

(mostly game controllers)



The Backends

- **SDL** – normalizes a lot of gamepads ()
- **evdev** – what Linux exposes and we can easily read on most devices
- **hidraw** – for some devices if we can read it

Normalized Form

- **Win32 is very HID-centric.**
- HID details leak in **DirectInput** (`device->GetObjectInfo(..., DIPH_BYUSAGE)`).
- **RawInput** allows to subscribe and receive RAWHID via messages sent to program's window.
- We normalize all controllers into HID internally and then consume / pass the HID representation in the user-facing APIs.

Normalized Form

```
BOOL hid_device_add_buttons(struct unix_device *iface, USAGE usage_page,
                           USAGE usage_min, USAGE usage_max)
{
    struct hid_report_descriptor *desc = &iface->hid_report_descriptor;
    const USHORT count = usage_max - usage_min + 1;
    const BYTE template[] =
    {
        USAGE_PAGE(2, usage_page),
        USAGE_MINIMUM(2, usage_min),
        USAGE_MAXIMUM(2, usage_max),
        LOGICAL_MINIMUM(1, 0),
        LOGICAL_MAXIMUM(1, 1),
        REPORT_COUNT(2, count),
        REPORT_SIZE(1, 1),
        INPUT(1, Data|Var|Abs),
    };

    /* ... */
}
```

Human Interface Devices

over USB and BT



Undoing Kernel's Work

- `hid_playstation` module
- puts the device into “**advanced mode**” right away (BT)
- this makes sense to get the **most of the device via evdev**

- stops sending well described input report #1
- starts sending **opaque** vendor report #49

We have to turn #49 into #1 until the game requests the fancy pants mode.

```
if ((impl->quirks & QUIRK_DUALSENSE_BT) && report_buffer[0] == 0x31 && size ≥ 11)
{
    BYTE trigger[2];
    size = 10;
    buff += 1;

    buff[0] = 1; /* fake report #1 */

    trigger[0] = buff[5]; /* TriggerLeft*/
    trigger[1] = buff[6]; /* TriggerRight */

    buff[5] = buff[8]; /* Buttons[0] */
    buff[6] = buff[9]; /* Buttons[1] */
    buff[7] = buff[10]; /* Buttons[2] */
    buff[8] = trigger[0]; /* TriggerLeft */
    buff[9] = trigger[1]; /* TirggerRight */
}
```

DualSense Haptics

- Extra **audio device** (over USB).
- **4 channels:**
 - 2 go to the 3.5mm jack
 - **2 go to the haptic motors**
- We are pairing the devices based on **sysfs paths**.
- Creating fake Container ID as there's no access to this device descriptor.

Access

Linux

```
% cat /dev/hidraw2  
cat: /dev/hidraw2: Permission denied
```

Windows

```
\\?\HID#VID_054C&PID_0CE6&MI_03#8&27cb3b47&0&000#{4d1e55b2-f16f-11cf-88cb-001111000030}  
  
HANDLE file = CreateFile(name, GENERIC_READ, SHARE_ALL, NULL, OPEN_EXISTING, 0, 0);  
ReadFile(file, buffer, sizeof(buffer), &bytes_read, NULL);
```

Mice, keyboards, pen devices, touchscreens and touchpads are opened exclusively by the OS.

```
% cat /usr/lib/udev/rules.d/70-steam-input.rules
```

```
...
```

```
# PS5 DualSense controller over USB hidraw
```

```
KERNEL="hidraw*", ATTRS{idVendor}="054c", ATTRS{idProduct}="0ce6", MODE="0660", TAG+="uaccess"
```

```
# PS5 DualSense controller over bluetooth hidraw
```

```
KERNEL="hidraw*", KERNELS="*054C:0CE6*", MODE="0660", TAG+="uaccess"
```

Users

- Wine / Proton
- Steam
- SDL
- ...

The Request

- **udev:** tag game controllers with **ID_GAME_CONTROLLER** (upstream)
- **udev:** **+uaccess** on all **ID_GAME_CONTROLLER** (either upstream or as an user-provided `.rules`)

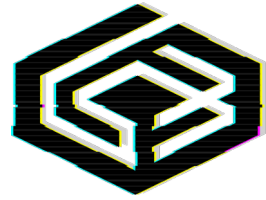
What We Are Getting

- **HIDIOCREVOKE IOCTL** in Linux 6.12
- **logind support** via `TakeDevice()`
- **Wayland protocol / XDG Desktop Portal** (?)
- **session managers** implementation of the portal using **logind's TakeDevice()** (?)
- wine support for the portal (?)

The Request

- **udev:** tag game controllers with **ID_GAME_CONTROLLER** (upstream)
- **udev:** **+uaccess** on all **ID_GAME_CONTROLLER** (either upstream or as an user-provided `.rules`)

Thanks!



CODE
WEAVERS®
SOFTWARE LIBERATORS

Questions?