Virtual Space Cadet Documentation

Release 0.1.0

Andrew Hardin

Apr 10, 2019

Contents

CHAPTER 1

How it works

The space cadet driver presents itself as a virtual device by intercepting and interpreting events from a physical device. The purpose of this document is to flesh out what that means.

1.1 Grab an input device

The space cadet driver starts by "grabbing" an input device, e.g. your keyboard device at `/dev/input/event4`. This is an exclusive grab - any events sent by the keyboard will be _intercepted_ by the driver.

1.2 State matrix

The driver polls the input device at a fixed frequency to check if any new events have occurred. Upon receiving an event (e.g. "`KC_A` was pressed"), the driver maps that event to a position in a 2D matrix. Like a physical keyboard's underlying firmware, this matrix records the binary state of every key (`{key_up = 0, key_down = 1}`).

The driver watches for changes in the state matrix. Key presses are detected when a state changes from *0 -> 1*. Releases are detected a state changes from *1 -> 0*. When a key press or release is detected, the _position_ of the change is used to determine which action to take (e.g. send *KC_A*). The mapping from state change to action is handled by a collection of layers.

1.3 Layers

The virtual keyboard's layout is composed of a series of layers. Each layer is a 2D matrix of key codes, and its dimension matches the driver's state matrix. Layers are stacked on top of one another - with higher layers taking precedence.

When the driver detects a change in the state matrix (e.g. `press @ {row 0, col 0}`), it loops through every enabled layer and forwards the event to the first non-transparent key it finds.

For example, the keyboard driver below has three 1x1 layers. When a `key_press` event is detected at `{row 0, col 0}`, the driver starts at the highest layer and goes down until it finds a non-transparent key - in this case `KC_A`.

```
(disabled) layer 1: [KC_B]          // candidate #1 skipped because layer disabled
(enabled)  layer 2: [TRANSPARENT]   // candidate #2 skipped because transparent
(enabled)  layer 0: [KC_A]          // candidate #3 accepted
```

For a better description of layers, please refer to QMK's discussion of layers. General concepts should transfer to this project.

1.4 Key Codes

What happens after an event is passed to a key code depends entirely on what type of key it is. In the simplest case, when a regular key code receives an event it immediately writes an event to the output device.

In addition to traditional keyboard codes, advanced codes such as macros, modifiers, and speed/tap sensitive keys can be included in a layer. The *key index* describes all possible keys.

CHAPTER 2

Quickstart

1. Install the project using Cargo. This will place the built binaries in the `~/.cargo` directory.

```
cargo install --git https://github.com/andrew-hardin/virtual-space-cadet.
↪git
```

2. Verify the binaries are findable. If this fails, the `~/.cargo` directory needs to be added to the `$PATH`.

```
which spacecadet
spacecadet --help
```

3. Download the demo matrix and layer files. The matrix file maps event codes to matrix locations. The layer file converts matrix locations to behaviors.

```
wget https://github.com/andrew-hardin/virtual-space-cadet/raw/master/
↪keyboards/vim_cursor/layers.json
wget https://github.com/andrew-hardin/virtual-space-cadet/raw/master/
↪keyboards/vim_cursor/matrix.json
```

3. Find your physical keyboard device under `/dev/input`. This isn't always easy task. Running `evtest` can sometimes be helpful in determining which device is your keyboard.

4. Attach the `spacecadet` driver to your keyboard device and remap the keys using the demo matrix and layer files.

```
spacecadet --device /dev/input/your-device \
           --layer layers.json \
           --matrix matrix.json
```

Tip: You may encounter permissions problems. The path of least resistance is to run the application as root.

An alternative long-term fix involves modifying permissions such that you can read the `/dev/input/your-device` and write to `/dev/uinput`.

---

5. With the `spacecadet` driver running, try typing on your physical keyboard - it should react normally. However, holding the space bar for longer than 150 milliseconds temporarily switches to a cursor layer. With space held, use `hjkl` to move the cursor left, down, up and right.

   For more information on this particular layout, read about the *Vim Cursor* layout.

CHAPTER 3

Keys

The space cadet driver supports a range of keys with advanced behavior, such as macros and layer switching.

3.1 Simple Keys

Simple keys are the normal keys you're used to typing, such as `KC_A`. The space cadet driver supports all the `EV_KEY` events that `libevdev` supports, a subset of which is enumerated below:

Code
KC_ESC
KC_1
KC_2
KC_3
KC_4
KC_5
KC_6
KC_7
KC_8
KC_9
KC_0
KC_Q
KC_W
KC_E
KC_R
KC_T
KC_Y
KC_U
KC_I
KC_O

Continued on next page

Table 1 – continued from previous page

KC_P
KC_A
KC_S
KC_D
KC_F
KC_G
KC_H
KC_J
KC_K
KC_L
KC_Z
KC_X
KC_C
KC_V
KC_B
KC_N
KC_M
KC_MINUS
KC_EQUAL
KC_BACKSPACE
KC_TAB
KC_LEFTBRACE
KC_RIGHTBRACE
KC_ENTER
KC_LEFTCTRL
KC_SEMICOLON
KC_APOSTROPHE
KC_GRAVE
KC_LEFTSHIFT
KC_BACKSLASH
KC_COMMA
KC_DOT
KC_SLASH
KC_RIGHTSHIFT
KC_KPASTERISK
KC_LEFTALT
KC_SPACE
KC_CAPSLOCK
KC_F1
KC_F2
KC_F3
KC_F4
KC_F5
KC_F6
KC_F7
KC_F8
KC_F9
KC_F10
KC_F11
KC_F12

Continued on next page

Table 1 – continued from previous page

KC_KP0
KC_KP1
KC_KP2
KC_KP3
KC_KP4
KC_KP5
KC_KP6
KC_KP7
KC_KP8
KC_KP9
KC_NUMLOCK
KC_SCROLLLOCK
KC_KPMINUS
KC_KPPLUS
KC_KPDOT
KC_ZENKAKUHANKAKU
KC_102ND
KC_RO
KC_KATAKANA
KC_HIRAGANA
KC_HENKAN
KC_KATAKANAHIRAGANA
KC_MUHENKAN
KC_KPJPCOMMA
KC_KPENTER
KC_RIGHTCTRL
KC_KPSLASH
KC_SYSRQ
KC_RIGHTALT
KC_LINEFEED
KC_HOME
KC_UP
KC_PAGEUP
KC_LEFT
KC_RIGHT
KC_END
KC_DOWN
KC_PAGEDOWN
KC_INSERT
KC_DELETE
KC_MACRO
KC_MUTE
KC_VOLUMEDOWN
KC_VOLUMEUP
KC_POWER
KC_KPEQUAL
KC_KPPLUSMINUS
KC_PAUSE
KC_SCALE
KC_KPCOMMA

Continued on next page

Table 1 – continued from previous page

KC_HANGEUL
KC_HANJA
KC_YEN
KC_LEFTMETA
KC_RIGHTMETA
KC_COMPOSE
KC_STOP
KC_AGAIN
KC_PROPS
KC_UNDO
KC_FRONT
KC_COPY
KC_OPEN
KC_PASTE
KC_FIND
KC_CUT
KC_HELP
KC_MENU
KC_CALC
KC_SETUP
KC_SLEEP
KC_WAKEUP
KC_FILE
KC_SENDFILE
KC_DELETEFILE
KC_XFER
KC_PROG1
KC_PROG2
KC_WWW
KC_MSDOS
KC_COFFEE
KC_ROTATE_DISPLAY
KC_CYCLEWINDOWS
KC_MAIL
KC_BOOKMARKS
KC_COMPUTER
KC_BACK
KC_FORWARD
KC_CLOSECD
KC_EJECTCD
KC_EJECTCLOSECD
KC_NEXTSONG
KC_PLAYPAUSE
KC_PREVIOUSSONG
KC_STOPCD
KC_RECORD
KC_REWIND
KC_PHONE
KC_ISO
KC_CONFIG

Continued on next page

Table 1 – continued from previous page

KC_HOMEPAGE
KC_REFRESH
KC_EXIT
KC_MOVE
KC_EDIT
KC_SCROLLUP
KC_SCROLLDOWN
KC_KPLEFTPAREN
KC_KPRIGHTPAREN
KC_NEW
KC_REDO
KC_F13
KC_F14
KC_F15
KC_F16
KC_F17
KC_F18
KC_F19
KC_F20
KC_F21
KC_F22
KC_F23
KC_F24
KC_PLAYCD
KC_PAUSECD
KC_PROG3
KC_PROG4
KC_DASHBOARD
KC_SUSPEND
KC_CLOSE
KC_PLAY
KC_FASTFORWARD
KC_BASSBOOST
KC_PRINT
KC_HP
KC_CAMERA
KC_SOUND
KC_QUESTION
KC_EMAIL
KC_CHAT
KC_SEARCH
KC_CONNECT
KC_FINANCE
KC_SPORT
KC_SHOP
KC_ALTERASE
KC_CANCEL
KC_BRIGHTNESSDOWN
KC_BRIGHTNESSUP
KC_MEDIA

Continued on next page

Table 1 – continued from previous page

KC_SWITCHVIDEOMODE
KC_KBDILLUMTOGGLE
KC_KBDILLUMDOWN
KC_KBDILLUMUP
KC_SEND
KC_REPLY
KC_FORWARDMAIL
KC_SAVE
KC_DOCUMENTS
KC_BATTERY
KC_BLUETOOTH
KC_WLAN
KC_UWB
KC_UNKNOWN
KC_VIDEO_NEXT
KC_VIDEO_PREV
KC_BRIGHTNESS_CYCLE
KC_BRIGHTNESS_AUTO
KC_DISPLAY_OFF
KC_WWAN
KC_RFKILL
KC_MICMUTE
KC_OK
KC_SELECT
KC_GOTO
KC_CLEAR
KC_POWER2
KC_OPTION
KC_INFO
KC_TIME
KC_VENDOR
KC_ARCHIVE
KC_PROGRAM
KC_CHANNEL
KC_FAVORITES
KC_EPG
KC_PVR
KC_MHP
KC_LANGUAGE
KC_TITLE
KC_SUBTITLE
KC_ANGLE
KC_ZOOM
KC_MODE
KC_KEYBOARD
KC_SCREEN
KC_PC
KC_TV
KC_TV2
KC_VCR

Continued on next page

Table 1 – continued from previous page

KC_VCR2
KC_SAT
KC_SAT2
KC_CD
KC_TAPE
KC_RADIO
KC_TUNER
KC_PLAYER
KC_TEXT
KC_DVD
KC_AUX
KC_MP3
KC_AUDIO
KC_VIDEO
KC_DIRECTORY
KC_LIST
KC_MEMO
KC_CALENDAR
KC_RED
KC_GREEN
KC_YELLOW
KC_BLUE
KC_CHANNELUP
KC_CHANNELDOWN
KC_FIRST
KC_LAST
KC_AB
KC_NEXT
KC_RESTART
KC_SLOW
KC_SHUFFLE
KC_BREAK
KC_PREVIOUS
KC_DIGITS
KC_TEEN
KC_TWEN
KC_VIDEOPHONE
KC_GAMES
KC_ZOOMIN
KC_ZOOMOUT
KC_ZOOMRESET
KC_WORDPROCESSOR
KC_EDITOR
KC_SPREADSHEET
KC_GRAPHICSEDITOR
KC_PRESENTATION
KC_DATABASE
KC_NEWS
KC_VOICEMAIL
KC_ADDRESSBOOK

Continued on next page

Table 1 – continued from previous page

KC_MESSENGER
KC_DISPLAYTOGGLE
KC_SPELLCHECK
KC_LOGOFF
KC_DOLLAR
KC_EURO
KC_FRAMEBACK
KC_FRAMEFORWARD
KC_CONTEXT_MENU
KC_MEDIA_REPEAT
KC_10CHANNELSUP
KC_10CHANNELSDOWN
KC_IMAGES
KC_DEL_EOL
KC_DEL_EOS
KC_INS_LINE
KC_DEL_LINE
KC_FN
KC_FN_ESC
KC_FN_F1
KC_FN_F2
KC_FN_F3
KC_FN_F4
KC_FN_F5
KC_FN_F6
KC_FN_F7
KC_FN_F8
KC_FN_F9
KC_FN_F10
KC_FN_F11
KC_FN_F12
KC_FN_1
KC_FN_2
KC_FN_D
KC_FN_E
KC_FN_F
KC_FN_S
KC_FN_B
KC_BRL_DOT1
KC_BRL_DOT2
KC_BRL_DOT3
KC_BRL_DOT4
KC_BRL_DOT5
KC_BRL_DOT6
KC_BRL_DOT7
KC_BRL_DOT8
KC_BRL_DOT9
KC_BRL_DOT10
KC_NUMERIC_0
KC_NUMERIC_1

Continued on next page

Table 1 – continued from previous page

KC_NUMERIC_2
KC_NUMERIC_3
KC_NUMERIC_4
KC_NUMERIC_5
KC_NUMERIC_6
KC_NUMERIC_7
KC_NUMERIC_8
KC_NUMERIC_9
KC_NUMERIC_STAR
KC_NUMERIC_POUND
KC_NUMERIC_A
KC_NUMERIC_B
KC_NUMERIC_C
KC_NUMERIC_D
KC_CAMERA_FOCUS
KC_WPS_BUTTON
KC_TOUCHPAD_TOGGLE
KC_TOUCHPAD_ON
KC_TOUCHPAD_OFF
KC_CAMERA_ZOOMIN
KC_CAMERA_ZOOMOUT
KC_CAMERA_UP
KC_CAMERA_DOWN
KC_CAMERA_LEFT
KC_CAMERA_RIGHT
KC_ATTENDANT_ON
KC_ATTENDANT_OFF
KC_ATTENDANT_TOGGLE
KC_LIGHTS_TOGGLE
KC_ALS_TOGGLE
KC_ROTATE_LOCK_TOGGLE
KC_BUTTONCONFIG
KC_TASKMANAGER
KC_JOURNAL
KC_CONTROLPANEL
KC_APPSELECT
KC_SCREENSAVER
KC_VOICECOMMAND
KC_ASSISTANT
KC_BRIGHTNESS_MIN
KC_BRIGHTNESS_MAX
KC_KBDINPUTASSIST_PREV
KC_KBDINPUTASSIST_NEXT
KC_KBDINPUTASSIST_PREVGROUP
KC_KBDINPUTASSIST_NEXTGROUP
KC_KBDINPUTASSIST_ACCEPT
KC_KBDINPUTASSIST_CANCEL
KC_RIGHT_UP
KC_RIGHT_DOWN
KC_LEFT_UP

Continued on next page

Table 1 – continued from previous page

KC_LEFT_DOWN
KC_ROOT_MENU
KC_MEDIA_TOP_MENU
KC_NUMERIC_11
KC_NUMERIC_12
KC_AUDIO_DESC
KC_3D_MODE
KC_NEXT_FAVORITE
KC_STOP_RECORD
KC_PAUSE_RECORD
KC_VOD
KC_UNMUTE
KC_FASTREVERSE
KC_SLOWREVERSE
KC_DATA
KC_ONSCREEN_KEYBOARD
KC_MAX

Warning: This table was created by copying and pasting the enums from evdev-rs. It probably should be created and maintained in an automated fashion.

3.2 Utility Keys

Utility key codes are used to fill space:

Key	Aliases	Description
`OPAQUE`	`X`, `XX`, `XXX`, ...	Ignore this key; a no-op.
`TRANSPARENT`	`_`, `___`, `____`, ...	Check the key in the next layer.

`OPAQUE` A black hole that swallows events but has no side effects. Useful when composing layers.

`TRANSPARENT` A key that allows and event to go through to the next lower layer. Useful when composing layers.

3.3 Advanced Keys

This is a catch-all category for keys that aren't simple and have nothing to do with layers.

Key	Description
`MACRO(key,...)`	Execute a macro of keys.
`WRAP(key_outer,key_inner)`	Wrap a key with another key, e.g. `WRAP(KC_LEFTSHIFT,KC_9)`.
`SPACECADET(key_tap, key_held)`	Emit different keys depending on whether the key is tapped or held.

`MACRO(...)` A macro key is a collection of keys that are pressed and released in order when the physical key is released.

`WRAP(OUTER,INNER)` Wrap an `INNER` key with an `OUTER` key. Useful for getting shifted characters such as `(){}`.

Example `WRAP(KC_LEFTSHIFT,KC_1)` -> !

`SPACECADET(KEY,HELD)` Emit a `KEY` when tapped, or act like `HELD` when held. This is similar to a one-shot-layer in the sense that key behavior depends on timing.

The specific use case for this key is modifying shifts to emit parentheses when tapped. This would be accomplished via:

Example `SPACECADET(WRAP(KC_LEFTSHIFT,KC_9),KC_LEFTSHIFT)`

3.4 Layer Keys

The layers of a space cadet driver can be manipulated using these keys:

Key	Description
`TG(layer)`	Toggle a layer.
`MO(layer)`	Momentarily enable a layer until the key is released.
`AL(layer)`	Activate a layer.
`LT(layer,key, hold_duration_ms)`	Enable a layer when held; emit a key when tapped.
`OSL(layer)`	Temporarily enable a layer until the next key is pressed + released.

`MACRO(...)` A macro key is a collection of keys that are pressed and released in order when the physical key is released.

`TG(LAYER)` Toggle whether a `LAYER` is turned on or off.

`MO(LAYER)` Enable a `LAYER` when the key is pressed, then disable the `LAYER` when the key is released.

`AL(LAYER)`: Activate a `LAYER`.

`OSL(LAYER)`: Enable a `LAYER` when pressed. The layer is disabled after another key is pressed and released. QMK calls this a "one-shot-layer" - it allows you to perform temporary layer switching without having to hold down a key.

`LT(LAYER,KEY,HOLD_DURATION_MS)`: Emit a `KEY` when tapped (i.e. pressed and released quickly). Enable a `LAYER` when held for more than `HOLD_DURATION_MS`. The layer is disabled when the held key is released.

CHAPTER 4

Keyboards

In a nod to QMK, the repository contains a handful of predefined keyboard layouts. The intent is that these can be used as design references when creating your own layout.

4.1 Simple Ergonomics

This is an example keyboard that performs simple key remapping to get slightly better ergonomics. The `CAPSLOCK` key doesn't deserve to be 1 key away from the home row...

Before	After
`KC_ESC`	`KC_CAPSLOCK`
`KC_TAB`	`KC_ESC`
`KC_CAPSLOCK`	`KC_TAB`

In addition to simple key remapping, the left and right shift keys are mapped to mapped to space cadet keys. The behavior of space cadet keys depends on whether the key is held or tapped:

	New Behavior	
Old Behavior	Held	Tapped
`LEFTSHIFT`	`LEFTSHIFT`	`(`
`RIGHTSHIFT`	`RIGHTSHIFT`	`)`

4.1.1 Layers

```
{
  "layer_order": [ "base" ],
  "base": {
    "enabled": true,
    "keys" : [
```

(continues on next page)

(continued from previous page)

```
      [ "KC_CAPSLOCK" ],
      [ "KC_ESC" ],
      [ "KC_TAB" ],
      [ "SPACECADET(WRAP(KC_LEFTSHIFT,KC_9),KC_LEFTSHIFT)" ],
      [ "SPACECADET(WRAP(KC_RIGHTSHIFT,KC_0),KC_RIGHTSHIFT)" ]
    ]
  }
}
```

4.1.2 Matrix

```
{
  "matrix": [
    [ "KC_ESC" ],
    [ "KC_TAB" ],
    [ "KC_CAPSLOCK" ],
    [ "KC_LEFTSHIFT" ],
    [ "KC_RIGHTSHIFT" ]
  ]
}
```

4.2 Vim Cursor

This is an example keyboard to demonstrate the power of layers and advanced key codes. The behaviour is simple:

- By default, keys act as expected.
- When the space bar is held, keys `HJKL` are remapped to move the cursor left, down, up, and right. The remapping is disabled when the space bar is released.

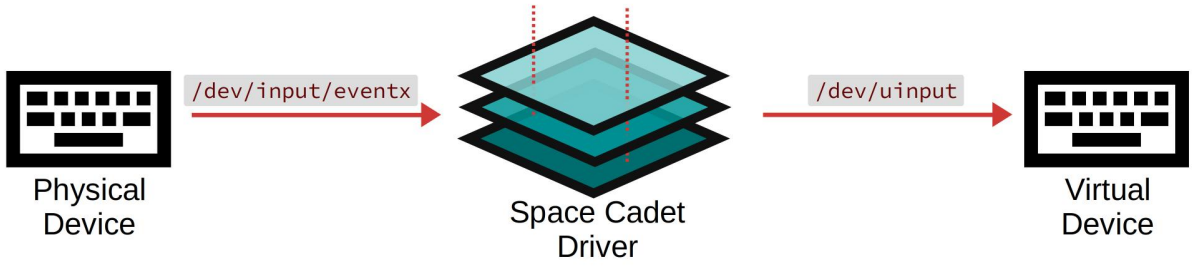
4.2.1 Layers

```
{
  "layer_order": [ "base", "cursor" ],
  "base": {
    "enabled": true,
    "keys" : [
      [ "KC_H", "KC_J", "KC_K", "KC_L" ],
      [ "LT(cursor,KC_SPACE,150)" ]
    ]
  },
  "cursor": {
    "enabled": false,
    "keys": [
      [ "KC_LEFT", "KC_DOWN", "KC_UP", "KC_RIGHT" ],
      [ "TRANSPARENT" ]
    ]
  }
}
```

4.2.2 Matrix

```
{
  "matrix": [
    [ "KC_H", "KC_J", "KC_K", "KC_L" ],
    [ "KC_SPACE" ]
  ]
}
```

This project creates a virtual keyboard device that supports layers and advanced key codes (inspired by QMK). The virtual keyboard device runs in user space - there's no need to modify the kernel. It intercepts events from an input device, interprets them, and emulates a new virtual keyboard device:



CHAPTER 5

Why does this exist?

The keyboard is my primary interface when working on a computer, but traditional keyboard drivers are painfully limited. I was overjoyed to discover the *flexibility* that came with running QMK firmware on my Kinesis Advantage2. However, running QMK for 10 months taught me two obvious lessons:

1. My laptop's keyboard doesn't support layers.
2. Secure facilities don't like it when you bring a custom keyboard to work.[1]

A laptop keyboard should be a first-class input device, and advanced capabilities should be available without needing to tweak the kernel.

Warning: Treat as a technical proof-of-concept - not something you can rely on for a decade. It meets my needs, although it's very rough around the edges (first time using Rust...).

[1] We trust keyboards running black-box firmware that was flashed onto the device in China, but only if the keyboard has `$large_manufacturer` printed on the box.

CHAPTER 6

Inspiration

- The spacefn-evdev project showed me that it was possible to emulate a keyboard layout in user space via `evdev` and `uinput`.
- The QMK project provided inspiration for layers and advanced key codes.

Index