libmk

Release 0.2.0

RedFantom

May 19, 2019

CONTENTS:

`libmk` and the Python wrapper `masterkeys` provide a low-level interface to the LEDs on the MasterKeys keyboard series.

CHAPTER

ONE

DEVICE SUPPORT

As Cooler Master Inc. has not provided any support, this library can only support a limited amount of devices, specifically, it can only support devices for which the record executable target has been executed. This program uses the library to register an offset for each key, which is required for the library to be able to control individual keys. Effects and full lighting colors can be set regardless of these offsets.

The current list of supported devices includes:

- MasterKeys Pro L RGB ANSI
- MasterKeys Pro S RGB ANSI (untested)
- MasterKeys Pro L RGB ISO (untested)

If you would like for your device to be supported as well, please run the `record` executable.

Keyboards with only monochrome lighting may use a different protocol and thus they would probably require more modifications than just adding a key layout matrix. Do not hesitate to open an issue if you have a monochrome keyboard, would like to see support and are willing to do some USB packet sniffing.

CHAPTER

TWO

EXAMPLES

In this section, a set of examples is provided as a simple reference on how the libraries may be used in a practical manner. The examples are extensively tested and a good way to determine whether building and installing the libraries was successful and the library is capable of controlling your device.

2.1 Photo Viewer

Simple Python program that reads a specified image file and calculates a color matrix based on this image by averaging the pixels. The image is then set on the keyboard.

```
"""
Author: RedFantom
License: GNU GPLv3
Copyright (c) 2018-2019 RedFantom
"""
import masterkeys as mk
from os import path
from PIL import Image
import sys


def print_progress(done, total):
    """Print a progress bar using #"""
    filled, percent = int(60 * done / total), done * 100 / total
    bar = "[{}]".format("#" * filled + " " * (60 - filled))
    sys.stdout.write("\r{} {:02.1f}%\r".format(bar, percent))
    sys.stdout.flush()


if __name__ == '__main__':
    """
    Open a file from a path given on input and process that to create a
    grid of key colors to set on the keyboard.
    """
    # Open the file
    file = input("File: ")
    if not path.exists(file):
        print("That file does not exist.")
        exit(-1)
    img = Image.open(file)

    # Process the image
```

(continues on next page)

(continued from previous page)

```
    w, h = img.size
    pixels = img.load()
    layout = mk.build_layout_list()
    w_p_c, h_p_r = (w // mk.MAX_COLS), (h // mk.MAX_ROWS)
    done, total = 0, mk.MAX_ROWS * mk.MAX_COLS
    for r in range(mk.MAX_ROWS):
        for c in range(mk.MAX_COLS):
            sums = [0, 0, 0]
            xrange = list(range(w_p_c * c, w_p_c * (c + 1)))
            yrange = list(range(h_p_r * r, h_p_r * (r + 1)))
            for x in xrange:
                for y in yrange:
                    for i in range(3):
                        sums[i] += pixels[x, y][i]
            color = (v // (w_p_c * h_p_r) for v in sums)
            color = tuple(map(int, color))
            layout[r][c] = color

            done += 1
            print_progress(done, total)
    print()

    # Update the color of the keyboard
    devices = mk.detect_devices()
    if len(devices) == 0:
        print("No devices connected.")
        exit(-2)
    mk.set_device(devices[0])
    mk.enable_control()
    mk.set_all_led_color(layout)
    mk.disable_control()
```

2.2 AmbiLight

Fast screenshot capture program that calculates the dominant color (average of colors with high hue) visible on the screen and sets it as the only color of the keyboard. Is capable of reaching somewhere between 20 and 30 FPS on most machines.

```
/**
 * Author: RedFantom
 * License: GNU GPLv3
 * Copyright (c) 2018-2019 RedFantom
*/
#include "libmk.h"
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <signal.h>
#include <pthread.h>
#include <time.h>
#include <X11/Xlib.h>
#include <X11/X.h>


```

(continues on next page)

(continued from previous page)

```
#define MAX_WIDTH -1  // 0: Full width, -1: / 2, n_pixels otherwise
#define SATURATION_BIAS 60
#define BRIGHTNESS_NORM
#define UPPER_TRESHOLD 700
#define LOWER_TRESHOLD 25


bool exit_requested = false;
unsigned char target_color[3] = {0};
pthread_mutex_t exit_req_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t target_color_lock = PTHREAD_MUTEX_INITIALIZER;
pthread_mutex_t keyboard_lock = PTHREAD_MUTEX_INITIALIZER;
Display* display;
Window root;
XWindowAttributes gwa;


typedef struct Screenshot {
    unsigned char* data;
    unsigned int width, height;
} Screenshot;


void interrupt_handler(int signal) {
    /** Handle a Control-C command to exit the loop */
    pthread_mutex_lock(&exit_req_lock);
    exit_requested = true;
    pthread_mutex_unlock(&exit_req_lock);
}


int capture_screenshot(Screenshot** screenshot) {
    /** Capture a screenshot and store it in an array */
    (*screenshot) = (Screenshot*) malloc(sizeof(Screenshot));
    int width = gwa.width, height = gwa.height;
    (*screenshot)->data = (unsigned char*) malloc(
        width * height * 3 * sizeof(unsigned char));

    XImage* img = XGetImage(
        display, root, 0, 0, width, height, AllPlanes, ZPixmap);

    unsigned long rm = img->red_mask, gm = img->green_mask, bm = img->blue_mask;
    unsigned long masks[3] = {rm, gm, bm};
    unsigned char pixel[3];

    for (int x=0; x < width; x++)
        for (int y = 0; y < height; y++) {
            unsigned long pix = XGetPixel(img, x, y);
            for (int i = 0; i  < 3; i++)
                (*screenshot)->data[(x + width * y) * 3 + i] =
                    (unsigned char) (pix >> (2 - i) * 8);
        }
    XDestroyImage(img);
    return 0;
}


```

(continues on next page)

(continued from previous page)

```
74  void* calculate_keyboard_color(void *void_ptr) {
75      /** Continuously capture screens and calculate the dominant colour
76       *
77       * Options are given in defines:
78       * MAX_WIDTH: If 0, uses full screen width. If -1, uses only the
79       *   first half of the pixel columns for double monitors. Otherwise,
80       *   limited to value of MAX_WIDTH.
81       * LOWER_THRESHOLD: Minimum summed value of the RGB triplet, used
82       *   for filtering out dark pixels.
83       * UPPER_THRESHOLD: Maximum summed value of the RGB triplet, used
84       *   for filtering out bright pixels.
85       * SATURATION_BIAS: Minimum required difference between any pair
86       *   of bytes of the RGB triplet of a pixel.
87       * BRIGHTNESS_NORM: If defined, target colors sent to the keyboard
88       *   are scaled so that at least one of the RGB values of the
89       *   triplet is the maximum of 255.
90       */
91      Screenshot* screen;
92
93      while (true) {
94
95          pthread_mutex_lock(&exit_req_lock);
96          if (exit_requested) {
97              printf("Exit requested.\n");
98              pthread_mutex_unlock(&exit_req_lock);
99              break;
100         }
101         pthread_mutex_unlock(&exit_req_lock);
102         if (capture_screenshot(&screen) < 0) {
103             int code = -2;
104             pthread_exit(&code);
105         }
106
107         int w = gwa.width, h = gwa.height;
108
109         unsigned char temp[3];
110         unsigned long colors[3] = {0};
111         unsigned long n_pixels = 0;
112         unsigned int sum;
113
114         int lim;
115         if (MAX_WIDTH == 0) {
116             lim = w;
117         } else if (MAX_WIDTH == -1) {
118             lim = w / 2;
119         } else {
120             lim = MAX_WIDTH;
121         }
122
123         // Sum
124         for (int x = 0; x < lim; x++) {
125             for (int y = 0; y < h; y++) {
126                 sum = 0;
127                 int max_diff = 0;
128                 for (int i = 0; i < 3; i++) {
129                     int first = i, second = i + 1;
130                     if (i == 2)
```

(continues on next page)

(continued from previous page)

```
                        second = 0;
                    int diff = screen->data[(x + y * w) * 3 + first] -
                               screen->data[(x + y * w) * 3 + second];
                    if (diff > max_diff)
                        max_diff = diff;

                    temp[i] = screen->data[(x + y * w) * 3 + i];
                    sum += temp[i];
                }
                if (sum < LOWER_TRESHOLD ||
                    sum > UPPER_TRESHOLD ||
                    max_diff < SATURATION_BIAS)
                    continue;
                for (int i = 0; i < 3; i++)
                    colors[i] += temp[i];
                n_pixels += 1;
            }
        }
        unsigned char color[3];
        unsigned char max = 0;

        // Average
        for (int i = 0; i < 3; i++) {
            if (n_pixels == 0) {
                color[i] = 0xFF;
                continue;
            }
            color[i] = (unsigned char) (colors[i] / n_pixels);
            if (color[i] > max)
                max = color[i];
        }

#ifdef BRIGHTNESS_NORM
        // Normalize
        if (max != 0)
            for (int i = 0; i < 3; i++)
                color[i] = (unsigned char) ((int) color[i] * (255.0 / max));
#endif

        // Copy color over to thread-safe variable
        pthread_mutex_lock(&target_color_lock);
        for (int i=0; i < 3; i++)
            target_color[i] = color[i];
        pthread_mutex_unlock(&target_color_lock);

        // Clean up
        free(screen->data);
        free(screen);
    }
    pthread_exit(0);
}


void* update_keyboard_color(void* ptr) {
    unsigned char color[3] = {0}, prev[3] = {0};
    while (true) {
        pthread_mutex_lock(&exit_req_lock);
```

(continues on next page)

(continued from previous page)

```
        if (exit_requested) {
            pthread_mutex_unlock(&exit_req_lock);
            break;
        }
        pthread_mutex_unlock(&exit_req_lock);

        int diff;
        bool equal = true;
        pthread_mutex_lock(&target_color_lock);
        for (int i=0; i < 3; i++) {
            diff = (int) target_color[i] - color[i];
            prev[i] = color[i];
            color[i] += (unsigned char) (diff / 20.0);
            equal = (prev[i] == target_color[i]) && equal;
        }
        pthread_mutex_unlock(&target_color_lock);

        if (equal)
            continue;

        pthread_mutex_lock(&keyboard_lock);
        int r = libmk_set_full_color(NULL, color[0], color[1], color[2]);
        if (r != LIBMK_SUCCESS)
            printf("LibMK Error: %d\n", r);
        pthread_mutex_unlock(&keyboard_lock);

        struct timespec time;
        time.tv_nsec = 100000000 / 4;
        nanosleep(&time, NULL);
    }
    pthread_exit(0);
}


int main() {
    /** Run a loop that grabs a screenshot and updated lighting */
    signal(SIGINT, interrupt_handler);
    libmk_init();

    // Set up libmk
    LibMK_Model* devices;
    int n = libmk_detect_devices(&devices);
    if (n < 0) {
        printf("libmk_detect_devices failed: %d\n", n);
        return n;
    }
    libmk_set_device(devices[0], NULL);
    libmk_enable_control(NULL);

    // Open the XDisplay
    display = XOpenDisplay(NULL);
    root = DefaultRootWindow(display);
    if (XGetWindowAttributes(display, root, &gwa) < 0)
        return -1;

    pthread_t keyboard, screenshot;

```

(continues on next page)

(continued from previous page)

```
245     // Run the loop
246     pthread_create(&screenshot, NULL, calculate_keyboard_color, NULL);
247     pthread_create(&keyboard, NULL, update_keyboard_color, NULL);
248 
249     pthread_join(screenshot, NULL);
250     pthread_join(keyboard, NULL);
251 
252     // Perform closing actions
253     libmk_disable_control(NULL);
254     libmk_exit();
255     return 0;
256 }
```

CHAPTER

THREE

DOCUMENTATION

3.1 Constants

libmk defines various constants that make interaction with the library easier. The following is a list of the definitions that can be used while interacting with the library.

LIBMK_MAX_ROWS
Maximum number of rows supported on any device.

LIBMK_MAX_COLS
Maximum number of columns supported on any device.

3.2 Enums

3.2.1 LibMK_ControlMode

enum LibMK_ControlMode
Supported control modes.

Values:

LIBMK_FIRMWARE_CTRL = 0x00
Default state, no interaction.

LIBMK_EFFECT_CTRL = 0x01
Software controls lighting effects.

LIBMK_CUSTOM_CTRL = 0x02
Software controls individual LEDs.

LIBMK_PROFILE_CTRL = 0x03
Software controls profiles.

3.2.2 LibMK_Effect

enum LibMK_Effect
LED Effect Types.

Values:

LIBMK_EFF_FULL = 0
All LEDs in a single color.

LIBMK_EFF_BREATH = 1
All LEDs single color turning slowly on and off.

LIBMK_EFF_BREATH_CYCLE = 2
All LEDs cycling through different colors.

LIBMK_EFF_SINGLE = 3
Keystrokes highlighted with fading light.

LIBMK_EFF_WAVE = 4
Color wave over all keys.

LIBMK_EFF_RIPPLE = 5
Rippling effect from keystroke.

LIBMK_EFF_CROSS = 6
Fading cross-effect from keystroke.

LIBMK_EFF_RAIN = 7
Diagonal streaks of light.

LIBMK_EFF_STAR = 8
Fading dots in a random pattern.

LIBMK_EFF_SNAKE = 9
Snake game.

LIBMK_EFF_CUSTOM = 10
Custom LED layout.

LIBMK_EFF_OFF = 0xFE
LEDs off.

LIBMK_EFF_SPECTRUM = 11
Not used.

LIBMK_EFF_RAPID_FIRE = 12
Not used.

3.2.3 LibMK_Layout

enum LibMK_Layout
Supported keyboard layouts.

Values:

LIBMK_LAYOUT_UNKNOWN = 0

LIBMK_LAYOUT_ANSI = 1

LIBMK_LAYOUT_ISO = 2

LIBMK_LAYOUT_JP = 3
Currently not supported.

3.2.4 LibMK_Model

enum LibMK_Model
Supported keyboard models.

Values:

DEV_RGB_L = 0

DEV_RGB_M = 5

DEV_RGB_S = 1

DEV_WHITE_L = 2

DEV_WHITE_M = 3

DEV_WHITE_S = 7

DEV_NOT_SET = -1
Device not set globally.

DEV_ANY = -2
Any supported device.

DEV_UNKNOWN = -3
Unrecognized device.

3.2.5 LibMK_Result

enum LibMK_Result
Error codes used within libmk.

Values:

LIBMK_SUCCESS = 0
The one and only success code.

LIBMK_ERR_INVALID_DEV = -1
Invalid device specified.

LIBMK_ERR_DEV_NOT_CONNECTED = -2
Device specified not connected.

LIBMK_ERR_DEV_NOT_SET = -3
Device has not been set.

LIBMK_ERR_UNKNOWN_LAYOUT = -14
Device has unknown layout.

LIBMK_ERR_DEV_NOT_CLOSED = -15
Device access not closed.

LIBMK_ERR_DEV_RESET_FAILED = -16
Device (libusb) reset failed.

LIBMK_ERR_IFACE_CLAIM_FAILED = -4
Failed to claim libusb interface.

LIBMK_ERR_IFACE_RELEASE_FAILED = -5
Failed to release libusb interface.

LIBMK_ERR_DEV_CLOSE_FAILED = -6
Failed to close libusb device.

LIBMK_ERR_DEV_OPEN_FAILED = -7
Failed to open libusb device.

LIBMK_ERR_KERNEL_DRIVER = -8
Failed to unload kernel driver.

LIBMK_ERR_DEV_LIST = -9
Failed to retrieve libusb device list.

LIBMK_ERR_TRANSFER = -10
Failed to transfer data to or from device.

LIBMK_ERR_DESCR = -11
Failed to get libusb device descriptor.

LIBMK_ERR_PROTOCOL = -13
Keyboard interaction protocol error.

LIBMK_ERR_INVALID_ARG = -14
Invalid arguments passed by caller.

LIBMK_ERR_STILL_ACTIVE = -15
Controller is still active.

3.2.6 LibMK_Size

enum LibMK_Size
Supported keyboard sizes.

Values:

LIBMK_L = 0

LIBMK_M = 1

LIBMK_S = 2

3.3 Functions

3.3.1 Library Control

bool **libmk_init** ( )
Initialize the library and its dependencies to a usable state.

Initializes a default libusb context for use throughout the library. If a call to this function is omitted, segmentation faults will be the result.

int **libmk_exit** ( )
Clean-up the library resources.

Frees up the memory used by the libusb context used throughout the library. Support for re-initialization after this function is called is not guaranteed. Use only at the end of the program.

3.3.2 Device Management

int **libmk_detect_devices** (*LibMK_Model* ***model_list*)
Search for devices and put the found models in an array.

Perform a search for devices using libusb and store all the found supported devices in an allocated array of LibMK_Model.

Return The number of found devices, or a LibMK_Result error code.

Parameters

- `model_list`: Pointer to an array of LibMK_Model enums. Required memory for storage is allocated by the function and must be freed by the caller.

LibMK_Device ***libmk_open_device**(libusb_device **device*)
Internal function. Loads the details of a device.

Loads the details of a USB device into a *LibMK_Device* struct instance. The details are used by libmk_detect_devices to determine whether a device is supported.

Return pointer to *LibMK_Device* instance if successful, NULL otherwise

Parameters

- `device`: libusb device descriptor to load details for

LibMK_Device ***libmk_create_device**(*LibMK_Model model*, libusb_device **device*, char **iManufacturer*, char **iProduct*, int *bVendor*, int *bDevice*)
Internal function. Allocate and fill *LibMK_Device* struct.

void **libmk_free_device**(*LibMK_Device *device*)
Internal function. Free memory of allocated *LibMK_Device*.

LibMK_Device ***libmk_append_device**(*LibMK_Device *first*, *LibMK_Device *device*)
Internal function. Build linked list with *LibMK_Device*.

LibMK_Model **libmk_ident_model**(char **product*)
Identify a model based on its USB descriptor product string.

3.3.3 Handles

int **libmk_create_handle**(*LibMK_Handle **handle*, *LibMK_Device *device*)
Internal function. Allocate and fill *LibMK_Handle* struct.

int **libmk_free_handle**(*LibMK_Handle *handle*)
Internal function. Free memory of allocated *LibMK_Handle*.

int **libmk_set_device**(*LibMK_Model model*, *LibMK_Handle **handle*)
Initialize a device within the library.

If NULL is passed for handle the function will set the global *LibMK_Handle* to the device specified. The function chooses the first available device of the specified model to assign to the handle.

Return LibMK_Result code, NULL or valid pointer in *handle.

Parameters

- `model`: Model to initialize. The model must be connected, else LIBMK_ERR_DEV_NOT_CONNECTED is returned.
- `handle`: Pointer to pointer of struct *LibMK_Handle*. The funtion allocates a *LibMK_Handle* struct and stores a pointer to it here.

3.3.4 Control

int **libmk_enable_control**(*LibMK_Handle *handle*)
Initialize the keyboard for control and send control packet.

Must be called in order to be able to control the keyboard. Claims the LED interface on the keyboard USB controller with the appropriate endpoints for control.

Return LibMK_Result result code

Parameters

- `handle`: LibMK_Handle* for the device to control. If NULL, the global handle is used.

int **libmk_disable_control**(*LibMK_Handle *handle*)
Release the keyboard from control.

Must be called when the user is done controlling the keyboard. Support for re-enabling of control on the same *LibMK_Handle* is not guaranteed.

Return LibMK_Result result code

Parameters

- `handle`: LibMK_Handle* for the device to release. If NULL, the global handle is used.

int **libmk_claim_interface**(*LibMK_Handle *handle*)
Internal function. Claims USB LED interface on device.

int **libmk_send_control_packet**(*LibMK_Handle *handle*)
Sends packet to put the keyboard in LIBMK_EFFECT_CTRL.

Return LibMK_Result result code

Parameters

- `handle`: LibMK_Handle* for the device to send the packet to. If NULL, the global handle is used.

int **libmk_reset**(*LibMK_Handle *handle*)
Internal function. Reset the libusb interface when releasing.

3.3.5 Communication

int **libmk_send_packet**(*LibMK_Handle *handle*, unsigned char **packet*)
Send a single packet and verify the response.

Sends a single packet of data to the keyboard of size LIBMK_PACKET_SIZE (will segfault if not properly sized!) and verifies the response of the keyboard by checking the header of the response packet. Frees the memory of the provided packet pointer.

Return LibMK_Result result code

Parameters

- `handle`: *LibMK_Handle* for the device to send the packet to
- `packet`: Array of bytes (unsigned char) to send of size LIBMK_PACKET_SIZE

int **libmk_exch_packet**(*LibMK_Handle *handle*, unsigned char **packet*)
Exchange a single packet with the keyboard.

Send a single packet to the keyboard and then store the response in the packet array. Does not free the packet memory or verify the response as an error response.

Return LibMK_Result result code, response in packet

Parameters

- `handle`: *LibMK_Handle* for the device to exchange data with
- `packet`: Array of bytes (unsigned char) to send of size LIBMK_PACKET_SIZE and to put response in

unsigned char ***libmk_build_packet**(unsigned char *predef*, ...)
Build a new packet of data that can be sent to keyboard.

Return Pointer to the allocated packet with the set bytes. NULL if no memory could be allocated.

Parameters

- `predef`: Amount of bytes given in the variable arguments
- `...`: Bytes to from index zero of the packet. The amount of bytes given must be equal to the amount specified by `predef`, otherwise this function will segfault.

3.3.6 LED Control

int **libmk_set_effect**(*LibMK_Handle* **handle*, *LibMK_Effect effect*)
Set effect to be active on keyboard.

Return LibMK_Result result code

Parameters

- `handle`: *LibMK_Handle* for device to set effect on. If NULL uses global device handle
- `effect`: LibMK_Effect specifier of effect to activate

int **libmk_set_effect_details**(*LibMK_Handle* **handle*, *LibMK_Effect_Details* **effect*)
Set effect to be active on keyboard with parameters.

Return LibMK_Result result code

Parameters

- `handle`: *LibMK_Handle* for device to set effect on. If NULL uses global device handle.
- `effect`: *LibMK_Effect_Details* instance with all parameters set

int **libmk_set_full_color**(*LibMK_Handle* **handle*, unsigned char *r*, unsigned char *g*, unsigned char *b*)
Set color of all the LEDs on the keyboard to a single color.

Return LibMK_Result result code

Parameters

- `handle`: *LibMK_Handle* for device to set color off. If NULL uses global device handle.
- `r`: color byte red
- `g`: color byte green
- `b`: color byte blue

int **libmk_set_all_led_color**(*LibMK_Handle* **handle*, unsigned char **colors*)
Set the color of all the LEDs on the keyboard individually.

Return LibMK_Result result code

Parameters

- `handle`: *LibMK_Handle* for device to set the key colors on. If NULL uses the global device handle.
- `colors`: Pointer to array of unsigned char of [LIBMK_MAX_ROWS][LIBMK_MAX_COLS][3] size.

int **libmk_set_single_led**(*LibMK_Handle* *handle, unsigned char *row*, unsigned char *col*, unsigned char *r*, unsigned char *g*, unsigned char *b*)

Set the color of a single LED on the keyboard.

Return LibMK_Result result code

Parameters

- `handle`: *LibMK_Handle* for device to set the color of the key on. If NULL uses the global device handle.
- `row`: Zero-indexed row index
- `col`: Zero-indexed column index
- `r`: color byte red
- `g`: color byte green
- `b`: color byte blue

int **libmk_get_offset**(unsigned char **offset*, *LibMK_Handle* *handle, unsigned char *row*, unsigned char *col*)

Retrieve the addressing offset of a specific key.

Return LibMK_Result result code

Parameters

- `offset`: Pointer to unsigned char to store offset in
- `handle`: *LibMK_Handle* for the device to find the offset for. Is required in order to determine the layout of the device.
- `row`: Zero-indexed row index
- `col`: Zero-indexed column index

3.4 Structs

3.4.1 LibMK_Firmware

struct LibMK_Firmware

Firmware details (version) and layout (speculative)

3.4.2 LibMK_Device

struct LibMK_Device

Struct describing a supported USB device.

This struct may be used as a linked list. Holds information required for the identification of the keyboard on the machine.

Public Members

char ***iManufacturer**
Manufacturer string.

char ***iProduct**
Product string.

int **bVendor**
USB Vendor ID number.

int **bDevice**
USB Device ID number.

LibMK_Model **model**
Model number.

struct *LibMK_Device* ***next**
Linked list attribute.

libusb_device ***device**
libusb_device struct instance

3.4.3 LibMK_Handle

struct LibMK_Handle
Struct describing an opened supported device.

Result of *libmk_set_device(LibMK_Model, LibMK_Handle**)*. Contains all the required data to control a keyboard device. Contains a libusb_device_handle* to allow interaction with the keyboard endpoints by the library. Multiple of these may be availble in your program, but no multiple handles for a single device are allowed.

Public Members

LibMK_Model **model**
Model of the keyboard this handle controls.

int **bVendor**
USB bVendor ID number.

int **bDevice**
USB bDevice ID number.

LibMK_Layout **layout**
Layout of this device.

LibMK_Size **size**
Size of this device.

libusb_device_handle ***handle**
libusb_device_handle required for reading from and writing to the device endpoints

bool **open**
Current state of the handle. If closed, the handle is no longer valid. Handles may not be re-opened.

3.4.4 LibMK_Effect_Details

struct LibMK_Effect_Details

Struct describing an effect with custom settings.

Apart from the default settings that are applied when an effect is applied using libmk_set_effect, custom settings may be applied to and effect. Each effect implements these settings differently.

For example: by using a background color of red and a foreground color orange with the effect LIBMK_EFF_STAR, a star effect is applied of fading orange-lit keys. All keys that are not 'stars' are red in color. The amount attribute changes the amount of 'stars' shown and the speed attribute the speed with which they fade in and out.

Not all attributes are supported by all effects. Some will have no influence at all.

Public Members

unsigned char **speed**
Running speed of the effect.

unsigned char **direction**
Direction of the effect.

unsigned char **amount**
Intensity modifier of the effect. Not supported by all effects in the same way.

unsigned char **foreground**[3]
Foreground color of the effect.

unsigned char **background**[3]
Background color of the effect.

CHAPTER

FOUR

DOCUMENTATION

4.1 Enums

enum LibMK_Controller_State
Controller States.

Author: RedFantom License: GNU GPLv3 Copyright (c) 2018 RedFantom

Values:

LIBMK_STATE_ACTIVE = 0
Controller is active.

LIBMK_STATE_STOPPED = 1
Controller was active, but is now stopped.

LIBMK_STATE_PRESTART = 2
Controller has not yet been active.

LIBMK_STATE_ERROR = 3
Controller was stopped by an error.

LIBMK_STATE_JOIN_ERR = 4
Failed to join the controller.

LIBMK_STATE_START_ERR = 5
Failed to start the controller.

enum LibMK_Instruction_Type
Instruction types.

Values:

LIBMK_INSTR_FULL = 0
Full keyboard color instruction.

LIBMK_INSTR_ALL = 1
All LEDs individually instruction.

LIBMK_INSTR_SINGLE = 2
Instruction for a single key.

4.2 Functions

LibMK_Controller ***libmk_create_controller**(*LibMK_Handle* *handle*)
Create a new *LibMK_Controller* for a defined handle.

After initialization of the Controller, the Handle may no longer be used directly. The lifecycle of the created controller is the responsibility of the user.

LibMK_Result **libmk_free_controller** (*LibMK_Controller* **c*)
Free a *LibMK_Controller*.

First performs a check to see if the controller is still active. An active controller may not be freed. Returns LIBMK_ERR_STILL_ACTIVE if the controller is still active.

int **libmk_sched_instruction** (*LibMK_Controller* **controller*, *LibMK_Instruction* **instruction*)
Schedule a linked-list of instructions.

Instruction scheduler than schedules the given linked-list of instructions at the end of the list of the controller in the given order. After scheduling, all instructions are given an ID number and they may not be scheduled again. After execution, the instructions are freed and thus after scheduling an instruction may not be accessed again.

Returns the instruction ID of the first instruction in the linked list (it is the user's responsibility to derive the ID number of the other instructions) upon success (postive integer) or a LibMK_Result (negative integer) upon failure.

LibMK_Result **libmk_cancel_instruction** (*LibMK_Controller* **c*, unsigned int *id*)
Cancel a scheduled instruction by its ID number.

If the instruction has already been executed, the instruction is not cancelled and the function fails quietly. Does not cancel any successive instructions even if the instruction was scheduled as part of a linked-list.

LibMK_Result **libmk_start_controller** (*LibMK_Controller* **controller*)
Start a new Controller thread.

Start the execution of instructions upon the keyboard in a different thread. This function enables control of the keyboard and initializes the thread.

void **libmk_run_controller** (*LibMK_Controller* **controller*)
Internal Function. Execute Controller instructions.

void **libmk_stop_controller** (*LibMK_Controller* **controller*)
Request an exit on a running controller.

The request is passed using the exit_flag, and thus the Controller may not immediately be stopped. To assure that the controller has stopped, use libmk_join_controller.

void **libmk_wait_controller** (*LibMK_Controller* **controller*)
Indicate the controller to finish only pending instructions.

If instructions are scheduled in the mean-time, they are added to the linked list and still executed by the Controller before exiting. Only after the linked list has become empty does the Controller exit.

This function sets the wait_flag, and thus the Controller has not necessarily stopped after this function ends. To assure that the Controller has stopped, use libmk_join_controller.

LibMK_Controller_State **libmk_join_controller** (*LibMK_Controller* **c*, double *t*)
Join the Controller thread.

Return LIBMK_STATE_JOIN_ERR upon timeout, controller state after exiting upon success.

Parameters

- `t`: Timeout in seconds

void **libmk_set_controller_error** (*LibMK_Controller* **c*, *LibMK_Result* *r*)
Internal Function.

LibMK_Instruction ***libmk_create_instruction**()
Allocate a new *LibMK_Instruction* struct.

LibMK_Instruction ***libmk_create_instruction_full**(unsigned char *c*[3])
Create a new instruction to set the full keyboard color.

Return : Pointer to single *LibMK_Instruction*. Duration may be set by the caller.

Parameters

- `c`: RGB color triplet that is copied to the instruction.

LibMK_Instruction ***libmk_create_instruction_all**(unsigned char *c*[*LIBMK_MAX_ROWS*][*LIBMK_MAX_COLS*][3])
Create a new instruction to set all leds individually.

Return : Pointer to single *LibMK_Instruction*. Duration may be set by the caller.

Parameters

- `c`: RGB color matrix that is copied to the instruction.

LibMK_Instruction ***libmk_create_instruction_flash**(unsigned char *c*[3], unsigned int *delay*, unsigned char *n*)
Create a new list of instructions to flash the keyboard.

Makes use of LIBMK_INSTR_FULL type instructions. Builds a linked list of n instructions.

Return : Pointer to linked list of *LibMK_Instruction*.

Parameters

- `c`: RGB color triplet that is copied to the instruction.
- `delay`: Duration set for each individual instruction of the linked list in microseconds, excluding the time required for instruction execution itself.
- `n`: Number of instructions in the linked list to be built.

LibMK_Instruction ***libmk_create_instruction_single**(unsigned char *row*, unsigned char *column*, unsigned char *c*[3])
: Create a new instruction to set the color of a single key

Overridden by a LIBMK_INSTR_FULL, just as in synchronous keyboard control. When changing six or more keys, using a LIBMK_INSTR_ALL is faster.

Return : Single *LibMK_Instruction*, duration may be set by the user.

Parameters

- `row`: Row coordinate of the key
- `column`: Column coordinate of the key
- `c`: RGB triplet to be copied to the instruction

void **libmk_free_instruction**(*LibMK_Instruction* **i*)
: Free a single *LibMK_Instruction*

The instruction is expected to longer be part of a linked list. This instruction is automatically called after an instruction has been executed and should only be called by the user if an instruction must be freed before it is scheduled. Use libmk_cancel_instruction to cancel scheduled instructions, which are then freed after cancelling.

LibMK_Result **libmk_exec_instruction** (*LibMK_Handle* *h, *LibMK_Instruction* *i)
: Internal Function. Execute a single instruction. NOT THREAD-SAFE.

4.3 Structs

struct LibMK_Instruction
Single instruction that can be executed by a controller.

An instruction should not be executed multiple times. An instruction is bound to a specific controller as its id attribute is bound to the linked list of instructions of a controller.

Public Members

unsigned char **c**
LIBMK_INSTR_SINGLE, row and column coords.

unsigned char ***colors**
LIBMK_INSTR_ALL, key color matrix.

unsigned char **color**[3]
LIBMK_INSTR_SINGLE, LIBMK_INSTR_FULL.

unsigned int **duration**
Delay after execution of instruction.

unsigned int **id**
ID number set by the scheduler.

struct *LibMK_Instruction* ***next**
Linked list attribute.

LibMK_Instruction_Type **type**
For the instruction execution.

struct LibMK_Controller
Controller for a keyboard managing a single handle.

Access to the various attributes of the Controller is protected by mutexes and the attributes of the controller should therefore not be accessed directly.

Public Members

LibMK_Handle ***handle**
Handle of the keyboard to control.

LibMK_Instruction ***instr**
Linked list of instructions.

pthread_mutex_t **instr_lock**
Protects LibMK_Instruction* instr.

pthread_t **thread**
Thread for libmk_run_controller.

pthread_mutex_t **exit_flag_lock**
Protects bool exit_flag and wait_flag.

bool **exit_flag**
: Exit event: Thread exits immediately.

bool **wait_flag**
: Wait event: Thread exits when all instructions are done.

pthread_mutex_t **state_lock**
: Protects LibMK_Controller_State state.

LibMK_Controller_State **state**
: Stores current state of controller.

pthread_mutex_t **error_lock**
: Protects LibMK_Result error.

LibMK_Result **error**
: Set for LIBMK_STATE_ERROR.

CHAPTER

FIVE

MASTERKEYS

Author: RedFantom License: GNU GPLv3 Copyright (c) 2018-2019 RedFantom

class masterkeys.**ControlMode**

    CUSTOM_CTRL = 2

    EFFECT_CTRL = 1

    FIRMWARE_CTRL = 0

    PROFILE_CTRL = 3

class masterkeys.**Effect**

    EFF_BREATH = 1

    EFF_BREATH_CYCLE = 2

    EFF_CROSS = 6

    EFF_FULL_ON = 0

    EFF_RAIN = 7

    EFF_RAPID_FIRE = 12

    EFF_REC = 10

    EFF_RIPPLE = 5

    EFF_SINGLE = 3

    EFF_SNAKE = 9

    EFF_SPECTRUM = 11

    EFF_STAR = 8

    EFF_WAVE = 4

class masterkeys.**Model**

    MODEL_ANY = -2

    MODEL_NOT_SET = -1

    MODEL_RGB_L = 0

    MODEL_RGB_M = 5

MODEL_RGB_S = 1

MODEL_UNKNOWN = -3

MODEL_WHITE_L = 2

MODEL_WHITE_M = 3

MODEL_WHITE_S = 7

class masterkeys.**ResultCode**

ERR_DESCR = -11

ERR_DEV_CLOSE_FAILED = -6

ERR_DEV_LIST = -9

ERR_DEV_NOT_CLOSED = -15

ERR_DEV_NOT_CONNECTED = -2

ERR_DEV_NOT_SET = -3

ERR_DEV_OPEN_FAILED = -7

ERR_DEV_RESET_FAILED = -16

ERR_IFACE_CLAIM_FAILED = -4

ERR_IFACE_RELEASE_FAILED = -5

ERR_INVALID_DEV = -1

ERR_KERNEL_DRIVER = -8

ERR_PROTOCOL = -13

ERR_TRANSFER = -10

ERR_UNKNOWN_LAYOUT = -14

SUCCESS = 0

masterkeys.**build_layout_list**()

Return a list of the right proportions for full LED layout

Returns Empty layout list

Return type List[List[Tuple[int, int, int], ..], ..]

masterkeys.**detect_devices**()

Detect supported connected devices and return a tuple of models

Returns Tuple of integers (*Model*)

Return type Tuple[int, ..]

Raises RuntimeError upon internal Python error

masterkeys.**disable_control**()

Disable control on the device that has been set and is controlled

Returns Result code (*ResultCode*)

Return type int

masterkeys.**enable_control**()

Enable control on the device that has been set

Returns Result code (*ResultCode*)

Return type int

masterkeys.**get_active_profile**()
Return the number of the profile active on the keyboard

Returns Result code (*ResultCode*) or profile number

Return type int

masterkeys.**get_device_ident**()
Return the bDevice USB descriptor value for the controlled keyboard

Returns bDevice USB descriptor value or result code (<0)

Return type int

masterkeys.**save_profile**()
Save the changes made to the lighting to the active profile

Returns Result code (*ResultCode*)

Return type int

masterkeys.**set_active_profile**(*profile*)
Activate a profile on the keyboard

Parameters **profile** (*int*) – Number of profile to activate

Returns Result code (*ResultCode*)

Return type int

masterkeys.**set_all_led_color**(*layout*)
Set the color of all LEDs on the keyboard individually

Parameters **layout** (*List[List[Tuple[int, int, int], ..], ..]*) – List of lists of color tuples such as created by build_layout_list()

Returns Result code (*ResultCode*)

Return type int

Raises ValueError if the wrong amount of elements is in the list

Raises TypeError if invalid argument type (any element)

masterkeys.**set_all_led_color_dict**(*keys*)
Set the color of all LEDs on the controlled device with a dictionary

The keys should be specified in a dictionary of the format {(row, col): (r, g, b)}

Parameters **keys** (*Dict[Tuple[int, int], Tuple[int, int, int]]*) – Dictionary containing key color data

Returns Result code (*ResultCode*)

Return type int

masterkeys.**set_control_mode**(*mode*)
Change the control mode of the keyboard manually

Parameters **mode** (*int*) – New control mode to set (*ControlMode*)

Returns Result code (*ResultCode*)

Return type int

masterkeys.**set_device**(*model*)
Set the device to be controlled by the library to the specified model

Parameters **model** (*int*) – Model to be controlled by the library. Only one device is supported in the Python library. Only the first found device of the specified model is controlled.

Returns Result code (*ResultCode*)

Return type int

Raises TypeError upon invalid argument type

masterkeys.**set_effect**(*effect*)
Set the effect to be active on the controlled keyboard

Parameters **effect** (*int*) – Effect number to set to be active (*Effect*)

Returns Result code (*ResultCode*)

Return type int

Raises TypeError upon invalid argument type

masterkeys.**set_effect_details**(*effect*, *direction=0*, *speed=96*, *amount=0*, *foreground=(255, 255, 255)*, *background=(0, 0, 0)*)
Set an effect with custom parameters

For more details about the parameters, please see the libmk documentation.

Parameters

- **effect** (*int*) – Effect number (*Effect*)
- **direction** (*int*) – Direction of the animation of the effect
- **speed** (*int*) – Speed of the animation of the effect
- **amount** (*int*) – Amount/intensity of the effect
- **foreground** (*Tuple[int, int, int]*) – Foreground color of the effect
- **background** (*Tuple[int, int, int]*) – Background color of the effect

Returns Result code (*ResultCode*)

Return type int

masterkeys.**set_full_led_color**(*r*, *g*, *b*)
Set the color of all the LEDs on the keyboard to a single color

Parameters

- **r** (*int*) – red color byte
- **g** (*int*) – green color byte
- **b** (*int*) – blue color byte

Returns Result code (*ResultCode*)

Return type int

masterkeys.**set_ind_led_color**(*row*, *col*, *r*, *g*, *b*)
Set the color of a single, individual key on the keyboard

Parameters

- **row** (*int*) – zero-indexed row index < MAX_ROWS
- **col** (*int*) – zero-indexed column index < MAX_COLS

- **r** (*int*) – red color byte
- **g** (*int*) – green color byte
- **b** (*int*) – blue color byte

Returns Result code (*ResultCode*)

Return type int

Raises `TypeError` upon invalid argument type

PYTHON MODULE INDEX

m

INDEX

M

P

R

S