
libmk

Release 0.3.0

RedFantom

Dec 16, 2020

CONTENTS:

1	Device Support	3
2	Examples	5
2.1	Photo Viewer	5
2.2	AmbiLight	6
3	Documentation	13
3.1	Constants	13
3.2	Enums	13
3.3	Functions	16
3.4	Structs	20
4	Documentation	23
4.1	Enums	23
4.2	Functions	24
4.3	Structs	26
5	masterkeys	29
	Python Module Index	35
	Index	37

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

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.

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.

```
1  """
2  Author: RedFantom
3  License: GNU GPLv3
4  Copyright (c) 2018-2019 RedFantom
5  """
6  import masterkeys as mk
7  from os import path
8  from PIL import Image
9  import sys
10
11
12  def print_progress(done, total):
13      """Print a progress bar using #"""
14      filled, percent = int(60 * done / total), done * 100 / total
15      bar = "[{}]" .format("#" * filled + " " * (60 - filled))
16      sys.stdout.write("\r{} {}{:02.1f}%\r" .format(bar, percent))
17      sys.stdout.flush()
18
19
20  if __name__ == '__main__':
21      """
22      Open a file from a path given on input and process that to create a
23      grid of key colors to set on the keyboard.
24      """
25      # Open the file
26      file = input("File: ")
27      if not path.exists(file):
28          print("That file does not exist.")
29          exit(-1)
30      img = Image.open(file)
31
32      # Process the image
```

(continues on next page)

(continued from previous page)

```

33     w, h = img.size
34     pixels = img.load()
35     layout = mk.build_layout_list()
36     w_p_c, h_p_r = (w // mk.MAX_COLS), (h // mk.MAX_ROWS)
37     done, total = 0, mk.MAX_ROWS * mk.MAX_COLS
38     for r in range(mk.MAX_ROWS):
39         for c in range(mk.MAX_COLS):
40             sums = [0, 0, 0]
41             xrange = list(range(w_p_c * c, w_p_c * (c + 1)))
42             yrange = list(range(h_p_r * r, h_p_r * (r + 1)))
43             for x in xrange:
44                 for y in yrange:
45                     for i in range(3):
46                         sums[i] += pixels[x, y][i]
47             color = (v // (w_p_c * h_p_r) for v in sums)
48             color = tuple(map(int, color))
49             layout[r][c] = color
50
51             done += 1
52             print_progress(done, total)
53     print()
54
55     # Update the color of the keyboard
56     devices = mk.detect_devices()
57     if len(devices) == 0:
58         print("No devices connected.")
59         exit(-2)
60     mk.set_device(devices[0])
61     mk.enable_control()
62     mk.set_all_led_color(layout)
63     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.

```

1  /**
2   * Author: RedFantom
3   * License: GNU GPLv3
4   * Copyright (c) 2018-2019 RedFantom
5  */
6  #include "libmk.h"
7  #include <stdbool.h>
8  #include <stdio.h>
9  #include <stdlib.h>
10 #include <signal.h>
11 #include <pthread.h>
12 #include <time.h>
13 #include <X11/Xlib.h>
14 #include <X11/X.h>
15
16

```

(continues on next page)

(continued from previous page)

```

17 #define MAX_WIDTH -1 // 0: Full width, -1: / 2, n_pixels otherwise
18 #define SATURATION_BIAS 60
19 #define BRIGHTNESS_NORM
20 #define UPPER_TRESHOLD 700
21 #define LOWER_TRESHOLD 25
22
23
24 bool exit_requested = false;
25 unsigned char target_color[3] = {0};
26 pthread_mutex_t exit_req_lock = PTHREAD_MUTEX_INITIALIZER;
27 pthread_mutex_t target_color_lock = PTHREAD_MUTEX_INITIALIZER;
28 pthread_mutex_t keyboard_lock = PTHREAD_MUTEX_INITIALIZER;
29 Display* display;
30 Window root;
31 XWindowAttributes gwa;
32
33
34 typedef struct Screenshot {
35     unsigned char* data;
36     unsigned int width, height;
37 } Screenshot;
38
39
40 void interrupt_handler(int signal) {
41     /** Handle a Control-C command to exit the loop */
42     pthread_mutex_lock(&exit_req_lock);
43     exit_requested = true;
44     pthread_mutex_unlock(&exit_req_lock);
45 }
46
47
48 int capture_screenshot(Screenshot** screenshot) {
49     /** Capture a Screenshot and store it in an array */
50     (*screenshot) = (Screenshot*) malloc(sizeof(Screenshot));
51     int width = gwa.width, height = gwa.height;
52     (*screenshot)->data = (unsigned char*) malloc(
53         width * height * 3 * sizeof(unsigned char));
54
55     XImage* img = XGetImage(
56         display, root, 0, 0, width, height, AllPlanes, ZPixmap);
57
58     unsigned long rm = img->red_mask, gm = img->green_mask, bm = img->blue_mask;
59     unsigned long masks[3] = {rm, gm, bm};
60     unsigned char pixel[3];
61
62     for (int x=0; x < width; x++)
63         for (int y = 0; y < height; y++) {
64             unsigned long pix = XGetPixel(img, x, y);
65             for (int i = 0; i < 3; i++)
66                 (*screenshot)->data[(x + width * y) * 3 + i] =
67                     (unsigned char) (pix >> (2 - i) * 8);
68         }
69     XDestroyImage(img);
70     return 0;
71 }
72
73

```

(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)

```

131         second = 0;
132         int diff = screen->data[(x + y * w) * 3 + first] -
133             screen->data[(x + y * w) * 3 + second];
134         if (diff > max_diff)
135             max_diff = diff;
136
137         temp[i] = screen->data[(x + y * w) * 3 + i];
138         sum += temp[i];
139     }
140     if (sum < LOWER_TRESHOLD ||
141         sum > UPPER_TRESHOLD ||
142         max_diff < SATURATION_BIAS)
143         continue;
144     for (int i = 0; i < 3; i++)
145         colors[i] += temp[i];
146     n_pixels += 1;
147 }
148 }
149 unsigned char color[3];
150 unsigned char max = 0;
151
152 // Average
153 for (int i = 0; i < 3; i++) {
154     if (n_pixels == 0) {
155         color[i] = 0xFF;
156         continue;
157     }
158     color[i] = (unsigned char) (colors[i] / n_pixels);
159     if (color[i] > max)
160         max = color[i];
161 }
162
163 #ifndef BRIGHTNESS_NORM
164     // Normalize
165     if (max != 0)
166         for (int i = 0; i < 3; i++)
167             color[i] = (unsigned char) ((int) color[i] * (255.0 / max));
168 #endif
169
170 // Copy color over to thread-safe variable
171 pthread_mutex_lock(&target_color_lock);
172 for (int i=0; i < 3; i++)
173     target_color[i] = color[i];
174 pthread_mutex_unlock(&target_color_lock);
175
176 // Clean up
177 free(screen->data);
178 free(screen);
179 }
180 pthread_exit(0);
181 }
182
183
184 void* update_keyboard_color(void* ptr) {
185     unsigned char color[3] = {0}, prev[3] = {0};
186     while (true) {
187         pthread_mutex_lock(&exit_req_lock);

```

(continues on next page)

(continued from previous page)

```

188     if (exit_requested) {
189         pthread_mutex_unlock(&exit_req_lock);
190         break;
191     }
192     pthread_mutex_unlock(&exit_req_lock);
193
194     int diff;
195     bool equal = true;
196     pthread_mutex_lock(&target_color_lock);
197     for (int i=0; i < 3; i++) {
198         diff = (int) target_color[i] - color[i];
199         prev[i] = color[i];
200         color[i] += (unsigned char) (diff / 20.0);
201         equal = (prev[i] == target_color[i]) && equal;
202     }
203     pthread_mutex_unlock(&target_color_lock);
204
205     if (equal)
206         continue;
207
208     pthread_mutex_lock(&keyboard_lock);
209     int r = libmk_set_full_color(NULL, color[0], color[1], color[2]);
210     if (r != LIBMK_SUCCESS)
211         printf("LibMK Error: %d\n", r);
212     pthread_mutex_unlock(&keyboard_lock);
213
214     struct timespec time;
215     time.tv_nsec = 100000000 / 4;
216     nanosleep(&time, NULL);
217 }
218 pthread_exit(0);
219 }
220
221
222 int main() {
223     /** Run a loop that grabs a screenshot and updated lighting */
224     signal(SIGINT, interrupt_handler);
225     libmk_init();
226
227     // Set up libmk
228     LibMK_Model* devices;
229     int n = libmk_detect_devices(&devices);
230     if (n < 0) {
231         printf("libmk_detect_devices failed: %d\n", n);
232         return n;
233     }
234     libmk_set_device(devices[0], NULL);
235     libmk_enable_control(NULL);
236
237     // Open the XDisplay
238     display = XOpenDisplay(NULL);
239     root = DefaultRootWindow(display);
240     if (XGetWindowAttributes(display, root, &gwa) < 0)
241         return -1;
242
243     pthread_t keyboard, screenshot;
244

```

(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 }
```


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:

enumerator LIBMK_FIRMWARE_CTRL

Default state, no interaction.

enumerator LIBMK_EFFECT_CTRL

Software controls lighting effects.

enumerator LIBMK_CUSTOM_CTRL

Software controls individual LEDs.

enumerator LIBMK_PROFILE_CTRL

Software controls profiles.

3.2.2 LibMK_Effect

enum LibMK_Effect

LED Effect Types.

Values:

enumerator LIBMK_EFF_FULL

All LEDs in a single color.

enumerator LIBMK_EFF_BREATH

All LEDs single color turning slowly on and off.

enumerator LIBMK_EFF_BREATH_CYCLE

All LEDs cycling through different colors.

enumerator LIBMK_EFF_SINGLE

Keystrokes highlighted with fading light.

enumerator LIBMK_EFF_WAVE

Color wave over all keys.

enumerator LIBMK_EFF_RIPPLE

Rippling effect from keystroke.

enumerator LIBMK_EFF_CROSS

Fading cross-effect from keystroke.

enumerator LIBMK_EFF_RAIN

Diagonal streaks of light.

enumerator LIBMK_EFF_STAR

Fading dots in a random pattern.

enumerator LIBMK_EFF_SNAKE

Snake game.

enumerator LIBMK_EFF_CUSTOM

Custom LED layout.

enumerator LIBMK_EFF_OFF

LEDs off.

enumerator LIBMK_EFF_SPECTRUM

Not used.

enumerator LIBMK_EFF_RAPID_FIRE

Not used.

3.2.3 LibMK_Layout

enum LibMK_Layout

Supported keyboard layouts.

Values:

enumerator LIBMK_LAYOUT_UNKNOWN

enumerator LIBMK_LAYOUT_ANSI

enumerator LIBMK_LAYOUT_ISO

enumerator LIBMK_LAYOUT_JP
Currently not supported.

3.2.4 LibMK_Model

enum LibMK_Model

Supported keyboard models.

Values:

enumerator DEV_RGB_L
enumerator DEV_RGB_M
enumerator DEV_RGB_S
enumerator DEV_WHITE_L
enumerator DEV_WHITE_M
enumerator DEV_WHITE_S
enumerator DEV_NOT_SET
Device not set globally.
enumerator DEV_ANY
Any supported device.
enumerator DEV_UNKNOWN
Unrecognized device.

3.2.5 LibMK_Result

enum LibMK_Result

Error codes used within libmk.

Values:

enumerator LIBMK_SUCCESS
The one and only success code.
enumerator LIBMK_ERR_INVALID_DEV
Invalid device specified.
enumerator LIBMK_ERR_DEV_NOT_CONNECTED
Device specified not connected.
enumerator LIBMK_ERR_DEV_NOT_SET
Device has not been set.
enumerator LIBMK_ERR_UNKNOWN_LAYOUT
Device has unknown layout.
enumerator LIBMK_ERR_DEV_NOT_CLOSED
Device access not closed.
enumerator LIBMK_ERR_DEV_RESET_FAILED
Device (libusb) reset failed.
enumerator LIBMK_ERR_IFACE_CLAIM_FAILED
Failed to claim libusb interface.

enumerator LIBMK_ERR_IFACE_RELEASE_FAILED

Failed to release libusb interface.

enumerator LIBMK_ERR_DEV_CLOSE_FAILED

Failed to close libusb device.

enumerator LIBMK_ERR_DEV_OPEN_FAILED

Failed to open libusb device.

enumerator LIBMK_ERR_KERNEL_DRIVER

Failed to unload kernel driver.

enumerator LIBMK_ERR_DEV_LIST

Failed to retrieve libusb device list.

enumerator LIBMK_ERR_TRANSFER

Failed to transfer data to or from device.

enumerator LIBMK_ERR_DESCR

Failed to get libusb device descriptor.

enumerator LIBMK_ERR_PROTOCOL

Keyboard interaction protocol error.

enumerator LIBMK_ERR_INVALID_ARG

Invalid arguments passed by caller.

enumerator LIBMK_ERR_STILL_ACTIVE

Controller is still active.

3.2.6 LibMK_Size

enum LibMK_Size

Supported keyboard sizes.

Values:

enumerator LIBMK_L

enumerator LIBMK_M

enumerator LIBMK_S

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 function 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 available 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 an 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.

DOCUMENTATION

4.1 Enums

enum LibMK_Controller_State

Controller States.

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

Values:

enumerator LIBMK_STATE_ACTIVE

Controller is active.

enumerator LIBMK_STATE_STOPPED

Controller was active, but is now stopped.

enumerator LIBMK_STATE_PRESTART

Controller has not yet been active.

enumerator LIBMK_STATE_ERROR

Controller was stopped by an error.

enumerator LIBMK_STATE_JOIN_ERR

Failed to join the controller.

enumerator LIBMK_STATE_START_ERR

Failed to start the controller.

enum LibMK_Instruction_Type

Instruction types.

Values:

enumerator LIBMK_INSTR_FULL

Full keyboard color instruction.

enumerator LIBMK_INSTR_ALL

All LEDs individually instruction.

enumerator LIBMK_INSTR_SINGLE

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 then 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 (positive 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.

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

masterkeys, [29](#)

B

`build_layout_list()` (in module *masterkeys*), 30

C

ControlMode (class in *masterkeys*), 29

CUSTOM_CTRL (*masterkeys.ControlMode* attribute), 29

D

`detect_devices()` (in module *masterkeys*), 30

`disable_control()` (in module *masterkeys*), 30

E

EFF_BREATH (*masterkeys.Effect* attribute), 29

EFF_BREATH_CYCLE (*masterkeys.Effect* attribute), 29

EFF_CROSS (*masterkeys.Effect* attribute), 29

EFF_FULL_ON (*masterkeys.Effect* attribute), 29

EFF_RAIN (*masterkeys.Effect* attribute), 29

EFF_RAPID_FIRE (*masterkeys.Effect* attribute), 29

EFF_REC (*masterkeys.Effect* attribute), 29

EFF_RIPPLE (*masterkeys.Effect* attribute), 29

EFF_SINGLE (*masterkeys.Effect* attribute), 29

EFF_SNAKE (*masterkeys.Effect* attribute), 29

EFF_SPECTRUM (*masterkeys.Effect* attribute), 29

EFF_STAR (*masterkeys.Effect* attribute), 29

EFF_WAVE (*masterkeys.Effect* attribute), 29

Effect (class in *masterkeys*), 29

EFFECT_CTRL (*masterkeys.ControlMode* attribute), 29

`enable_control()` (in module *masterkeys*), 30

ERR_DESCR (*masterkeys.ResultCode* attribute), 30

ERR_DEV_CLOSE_FAILED (*masterkeys.ResultCode* attribute), 30

ERR_DEV_LIST (*masterkeys.ResultCode* attribute), 30

ERR_DEV_NOT_CLOSED (*masterkeys.ResultCode* attribute), 30

ERR_DEV_NOT_CONNECTED (*masterkeys.ResultCode* attribute), 30

ERR_DEV_NOT_SET (*masterkeys.ResultCode* attribute), 30

ERR_DEV_OPEN_FAILED (*masterkeys.ResultCode* attribute), 30

ERR_DEV_RESET_FAILED (*masterkeys.ResultCode* attribute), 30

ERR_IFACE_CLAIM_FAILED (*masterkeys.ResultCode* attribute), 30

ERR_IFACE_RELEASE_FAILED (*masterkeys.ResultCode* attribute), 30

ERR_INVALID_DEV (*masterkeys.ResultCode* attribute), 30

ERR_KERNEL_DRIVER (*masterkeys.ResultCode* attribute), 30

ERR_PROTOCOL (*masterkeys.ResultCode* attribute), 30

ERR_TRANSFER (*masterkeys.ResultCode* attribute), 30

ERR_UNKNOWN_LAYOUT (*masterkeys.ResultCode* attribute), 30

F

FIRMWARE_CTRL (*masterkeys.ControlMode* attribute), 29

G

`get_active_profile()` (in module *masterkeys*), 31

`get_device_ident()` (in module *masterkeys*), 31

L

libmk_append_device (C++ function), 17

libmk_build_packet (C++ function), 19

libmk_cancel_instruction (C++ function), 24

libmk_claim_interface (C++ function), 18

LibMK_Controller (C++ struct), 26

LibMK_Controller::error (C++ member), 27

LibMK_Controller::error_lock (C++ member), 27

LibMK_Controller::exit_flag (C++ member), 27

LibMK_Controller::exit_flag_lock (C++ member), 27

LibMK_Controller::handle (C++ member), 27

LibMK_Controller::instr (C++ member), 27

LibMK_Controller::instr_lock (C++ member), 27

LibMK_Controller::state (C++ member), 27

LibMK_Controller::state_lock (C++ member), 27

LibMK_Controller::thread (C++ member), 27

LibMK_Controller::wait_flag (C++ member), 27
 LibMK_Controller_State (C++ enum), 23
 LibMK_Controller_State::LIBMK_STATE_ACTIVE (C++ enumerator), 23
 LibMK_Controller_State::LIBMK_STATE_ERROR (C++ enumerator), 23
 LibMK_Controller_State::LIBMK_STATE_JOIN_ERR (C++ enumerator), 23
 LibMK_Controller_State::LIBMK_STATE_PRESTART (C++ enumerator), 23
 LibMK_Controller_State::LIBMK_STATE_START_ERR (C++ enumerator), 23
 LibMK_Controller_State::LIBMK_STATE_STOPPED (C++ enumerator), 23
 LibMK_ControlMode (C++ enum), 13
 LibMK_ControlMode::LIBMK_CUSTOM_CTRL (C++ enumerator), 13
 LibMK_ControlMode::LIBMK_EFFECT_CTRL (C++ enumerator), 13
 LibMK_ControlMode::LIBMK_FIRMWARE_CTRL (C++ enumerator), 13
 LibMK_ControlMode::LIBMK_PROFILE_CTRL (C++ enumerator), 13
 libmk_create_controller (C++ function), 24
 libmk_create_device (C++ function), 17
 libmk_create_handle (C++ function), 17
 libmk_create_instruction (C++ function), 25
 libmk_create_instruction_all (C++ function), 25
 libmk_create_instruction_flash (C++ function), 25
 libmk_create_instruction_full (C++ function), 25
 libmk_create_instruction_single (C++ function), 25
 libmk_detect_devices (C++ function), 17
 LibMK_Device (C++ struct), 21
 LibMK_Device::bDevice (C++ member), 21
 LibMK_Device::bVendor (C++ member), 21
 LibMK_Device::device (C++ member), 21
 LibMK_Device::iManufacturer (C++ member), 21
 LibMK_Device::iProduct (C++ member), 21
 LibMK_Device::model (C++ member), 21
 LibMK_Device::next (C++ member), 21
 libmk_disable_control (C++ function), 18
 LibMK_Effect (C++ enum), 14
 LibMK_Effect::LIBMK_EFF_BREATH (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_BREATH_CYCLE (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_CROSS (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_CUSTOM (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_FULL (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_OFF (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_RAIN (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_RAPID_FIRE (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_RIPPLE (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_SINGLE (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_SNAKE (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_SPECTRUM (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_STAR (C++ enumerator), 14
 LibMK_Effect::LIBMK_EFF_WAVE (C++ enumerator), 14
 LibMK_Effect_Details (C++ struct), 22
 LibMK_Effect_Details::amount (C++ member), 22
 LibMK_Effect_Details::background (C++ member), 22
 LibMK_Effect_Details::direction (C++ member), 22
 LibMK_Effect_Details::foreground (C++ member), 22
 LibMK_Effect_Details::speed (C++ member), 22
 libmk_enable_control (C++ function), 18
 libmk_exch_packet (C++ function), 18
 libmk_exec_instruction (C++ function), 26
 libmk_exit (C++ function), 16
 LibMK_Firmware (C++ struct), 20
 libmk_free_controller (C++ function), 24
 libmk_free_device (C++ function), 17
 libmk_free_handle (C++ function), 17
 libmk_free_instruction (C++ function), 25
 libmk_get_offset (C++ function), 20
 LibMK_Handle (C++ struct), 21
 LibMK_Handle::bDevice (C++ member), 21
 LibMK_Handle::bVendor (C++ member), 21
 LibMK_Handle::handle (C++ member), 21
 LibMK_Handle::layout (C++ member), 21
 LibMK_Handle::model (C++ member), 21
 LibMK_Handle::open (C++ member), 22
 LibMK_Handle::size (C++ member), 21
 libmk_ident_model (C++ function), 17
 libmk_init (C++ function), 16
 LibMK_Instruction (C++ struct), 26

LibMK_Instruction::c (C++ member), 26
 LibMK_Instruction::color (C++ member), 26
 LibMK_Instruction::colors (C++ member), 26
 LibMK_Instruction::duration (C++ member), 26
 LibMK_Instruction::id (C++ member), 26
 LibMK_Instruction::next (C++ member), 26
 LibMK_Instruction::type (C++ member), 26
 LibMK_Instruction_Type (C++ enum), 23
 LibMK_Instruction_Type::LIBMK_INSTR_ALL (C++ enumerator), 23
 LibMK_Instruction_Type::LIBMK_INSTR_FULLL (C++ enumerator), 23
 LibMK_Instruction_Type::LIBMK_INSTR_SINGLE (C++ enumerator), 23
 libmk_join_controller (C++ function), 24
 LibMK_Layout (C++ enum), 14
 LibMK_Layout::LIBMK_LAYOUT_ANSI (C++ enumerator), 14
 LibMK_Layout::LIBMK_LAYOUT_ISO (C++ enumerator), 14
 LibMK_Layout::LIBMK_LAYOUT_JP (C++ enumerator), 14
 LibMK_Layout::LIBMK_LAYOUT_UNKNOWN (C++ enumerator), 14
 LIBMK_MAX_COLS (C macro), 13
 LIBMK_MAX_ROWS (C macro), 13
 LibMK_Model (C++ enum), 15
 LibMK_Model::DEV_ANY (C++ enumerator), 15
 LibMK_Model::DEV_NOT_SET (C++ enumerator), 15
 LibMK_Model::DEV_RGB_L (C++ enumerator), 15
 LibMK_Model::DEV_RGB_M (C++ enumerator), 15
 LibMK_Model::DEV_RGB_S (C++ enumerator), 15
 LibMK_Model::DEV_UNKNOWN (C++ enumerator), 15
 LibMK_Model::DEV_WHITE_L (C++ enumerator), 15
 LibMK_Model::DEV_WHITE_M (C++ enumerator), 15
 LibMK_Model::DEV_WHITE_S (C++ enumerator), 15
 libmk_open_device (C++ function), 17
 libmk_reset (C++ function), 18
 LibMK_Result (C++ enum), 15
 LibMK_Result::LIBMK_ERR_DESCR (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_DEV_CLOSE_FAILED (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_DEV_LIST (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_DEV_NOT_CLOSED (C++ enumerator), 15
 LibMK_Result::LIBMK_ERR_DEV_NOT_CONNECTED (C++ enumerator), 15
 LibMK_Result::LIBMK_ERR_DEV_NOT_SET (C++ enumerator), 15
 LibMK_Result::LIBMK_ERR_DEV_OPEN_FAILED (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_DEV_RESET_FAILED (C++ enumerator), 15
 LibMK_Result::LIBMK_ERR_IFACE_CLAIM_FAILED (C++ enumerator), 15
 LibMK_Result::LIBMK_ERR_IFACE_RELEASE_FAILED (C++ enumerator), 15
 LibMK_Result::LIBMK_ERR_INVALID_ARG (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_INVALID_DEV (C++ enumerator), 15
 LibMK_Result::LIBMK_ERR_KERNEL_DRIVER (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_PROTOCOL (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_STILL_ACTIVE (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_TRANSFER (C++ enumerator), 16
 LibMK_Result::LIBMK_ERR_UNKNOWN_LAYOUT (C++ enumerator), 15
 LibMK_Result::LIBMK_SUCCESS (C++ enumerator), 15
 libmk_run_controller (C++ function), 24
 libmk_sched_instruction (C++ function), 24
 libmk_send_control_packet (C++ function), 18
 libmk_send_packet (C++ function), 18
 libmk_set_all_led_color (C++ function), 19
 libmk_set_controller_error (C++ function), 25
 libmk_set_device (C++ function), 17
 libmk_set_effect (C++ function), 19
 libmk_set_effect_details (C++ function), 19
 libmk_set_full_color (C++ function), 19
 libmk_set_single_led (C++ function), 20
 LibMK_Size (C++ enum), 16
 LibMK_Size::LIBMK_L (C++ enumerator), 16
 LibMK_Size::LIBMK_M (C++ enumerator), 16
 LibMK_Size::LIBMK_S (C++ enumerator), 16
 libmk_start_controller (C++ function), 24
 libmk_stop_controller (C++ function), 24
 libmk_wait_controller (C++ function), 24

M

masterkeys module, 29
 Model (class in masterkeys), 29
 MODEL_ANY (masterkeys.Model attribute), 29
 MODEL_NOT_SET (masterkeys.Model attribute), 29
 MODEL_RGB_L (masterkeys.Model attribute), 29

MODEL_RGB_M (*masterkeys.Model attribute*), 29
MODEL_RGB_S (*masterkeys.Model attribute*), 29
MODEL_UNKNOWN (*masterkeys.Model attribute*), 30
MODEL_WHITE_L (*masterkeys.Model attribute*), 30
MODEL_WHITE_M (*masterkeys.Model attribute*), 30
MODEL_WHITE_S (*masterkeys.Model attribute*), 30
module
 masterkeys, 29

P

PROFILE_CTRL (*masterkeys.ControlMode attribute*),
29

R

ResultCode (*class in masterkeys*), 30

S

save_profile() (*in module masterkeys*), 31
set_active_profile() (*in module masterkeys*), 31
set_all_led_color() (*in module masterkeys*), 31
set_all_led_color_dict() (*in module masterkeys*), 31
set_control_mode() (*in module masterkeys*), 31
set_device() (*in module masterkeys*), 31
set_effect() (*in module masterkeys*), 32
set_effect_details() (*in module masterkeys*), 32
set_full_led_color() (*in module masterkeys*), 32
set_ind_led_color() (*in module masterkeys*), 32
SUCCESS (*masterkeys.ResultCode attribute*), 30