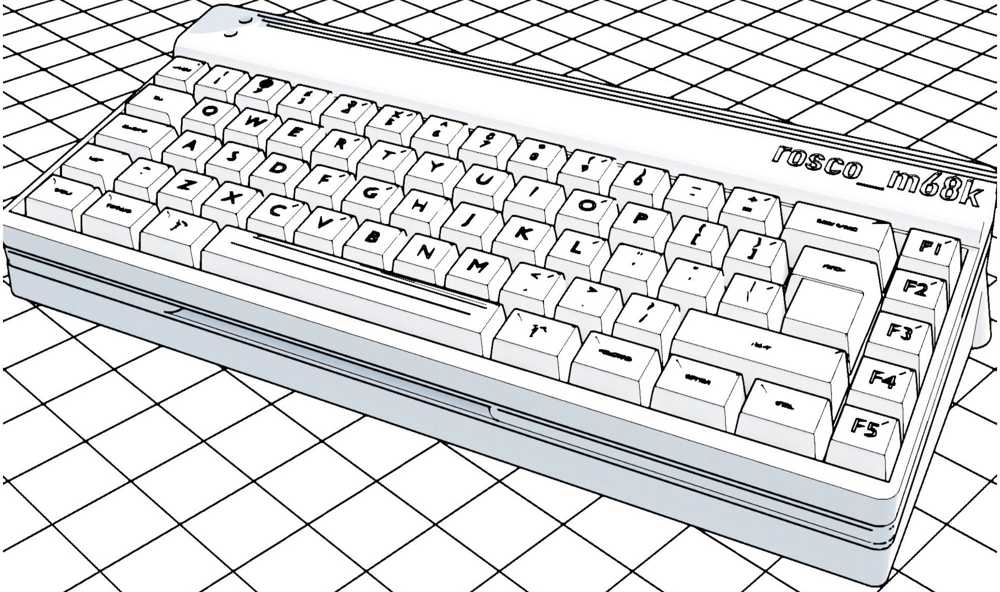
rosco_m68k



KEYBOARD USER'S MANUAL

Thank you For Your Purchase!

Thank you for purchasing the rosco_m68k® keyboard - we're sure you'll be delighted with your new retro-and-homebrew-focused fully mechanical keyboard!

This manual contains important information about using your new keyboard – please read it carefully and keep it handy so you can refer to it later.

By following the instructions in this manual, your rosco_m68k Keyboard will be easy to use and should give you years of trouble-free operation.

In the unlikely event that you experience any trouble or need some help, do not hesitate to contact us at hello@rosco-m68k.com.

Join the community on Discord: https://discord.gg/3efKTfW2NZ

Table of Contents

1. Important Safety Notices

Due to small parts presenting a choking hazard, this kit is not suitable for small children.

Exercise caution and follow all equipment manufacturer's instructions when building your kit – some equipment may cause harm and serious injury if not used correctly.

Ensure power is not applied to your board during assembly. Always follow the instructions from your power supply manufacturer.

Your keyboard is designed to operate at 3.3V - 5V DC. **Do not apply voltages outside of this range or malfunction and/or permanent damage may result. Excessive voltages will present a risk of fire.**

Observe all warnings and precautions recommended by the manufacturer of all solders, fluxes and other chemicals during assembly.

If using isopropyl alcohol to clean your board post-assembly, closely follow the manufacturer's instructions. **This chemical is highly poisonous and can be absorbed through skin contact.**

Do not allow children or pets in the area during assembly due to risks from equipment and chemicals used.

Observe all local environmental and other regulations when using or disposing of components, waste equipment, tooling and chemicals.

2. General Description

The rosco_m68k® Keyboard is a 67-key all-mechanical keyboard for the rosco_m68k and other minimal computer systems.

Designed specifically for retro and homebrew computers, the rosco_m68k Keyboard uses a TTL serial interface and can operate in various modes to enable easy use with your rosco_m68k® or other retro / homebrew computer.

Features

- 67 plate-mount keys with Cherry-MX-compatible 5-pin switches
- Full N-Key Rollover (NKRO)
- ASCII and Scancode modes for maximum flexibility
- TTL Serial interface supporting 115200 or 9600 bps
- 3.3V and 5V operation for maximum compatibility**
- Cherry-style PCB mount stabilizers
- Command interface for enhanced software control
- 3-axis, 3-button PS-2 mouse interface*
- SPI & LED extension interfaces*

* Selected products only
** 3.3V operation requires compatible peripherals

3. Switches, Stabilizers & Keycaps

If you purchased your keyboard PCB-only, this section contains some specifications on the additional hardware you will need to source.

The keyboard is designed for use with Cherry MX-compatible switches - we recommend Gateron KS-9-CAP 5-pin switches. We also recommend that 3-pin switches are avoided unless a plate is to be used for strength and stability.

The board directly supports Cherry-compatible screw-in PCB mount stabilizers – we test with genuine Cherry hardware, which is commonly available online. You will require:

1 x 6.25u Space bar wire
3 x 2u modifier wires
8 x Stabiliser housings and stems
8 x Screws

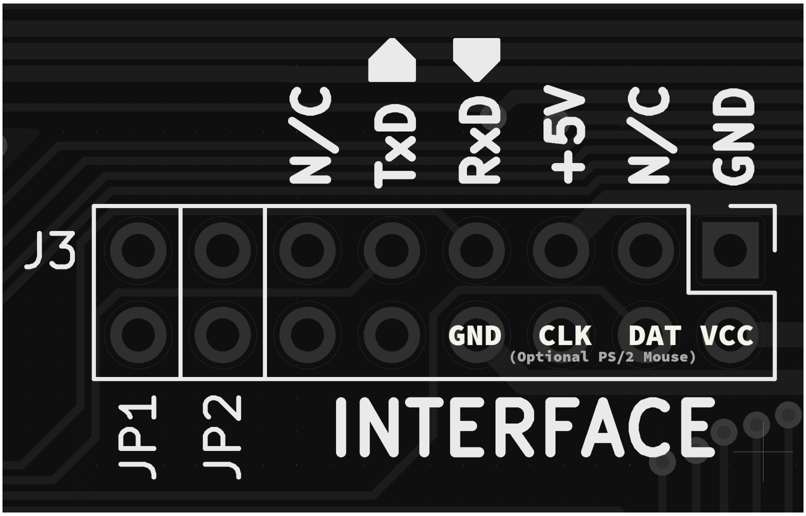
Any commodity Cherry-compatible keycap set should work fine, assuming it contains the right number of keys of the appropriate size. We've found that 104-key PC sets work well, and are available from Amazon – though you may find you need to sub a 1u key for one of the 1.25u (in which case we tend to swap out right-command).

For keycaps, you'll need Cherry MX compatible, and many options are available. Personally we like OEM (which does mean a few keys "stick out") but if you want a uniform look, you might prefer DSA or XDA profile.

Check out https://switchandclick.com/sa-vs-dsa-vs-oem-vs-cherry-vs-xda-keycap-profiles/ for more information on the different profile options – and follow links to find out more about switches, stabilizers, and all things Keyboard!

4. Connection to rosco_m68k

To connect the keyboard to your rosco_m68k, the following pins on the INTERFACE connector are used:



This pinout matches the pinout on the rosco_m68k main board, and also on commonly-available UART adaptors. We recommend mounting a right-angle 01x06 male header on the keyboard and using a straight six-way cable with DuPont connectors.

Connect to UART B on the rosco_m68k. Firmware 2.42 or higher is required for autodetection.

You will need to ensure the 'POWER' jumper (JP2) on the rosco_m68k is shorted, or your keyboard will receive no power!

5. Jumper Settings

The board has several jumpers for advanced use-cases:

JP1 – Speed Selection

Open	115,200bps
Short	9600bps

JP2 – Start-up Mode Selection

Open	ASCII Mode
Short	Scancode Mode

JP3 – Passthrough Mode (See below)

Open	No Passthrough
Short	Passthrough Enabled

JP4 – PS/2 Mouse

Open	Mouse enabled (see info panel below)
Short	Mouse disabled

JP5 – Do Not Short

This jumper is used for testing at the factory, may cause malfunction if shorted.

JP6 – Do Not Short

This jumper is used for testing at the factory, will cause malfunction if shorted.

The mouse will not automatically send reports by default, and will not be initialized until MOUSE_DETECT, MOUSE_STRM_ON or MOUSE_REPORT commands are sent.

Streaming (with MOUSE_STRM_ON) is only supported in Scancode mode – but MOUSE_REPORT can be used in any mode.

6. General Operation

The keyboard can operate in two modes – ASCII mode and Scancode mode. The start-up mode is selected via JP2, and can be changed at runtime with the MODE_SET command.

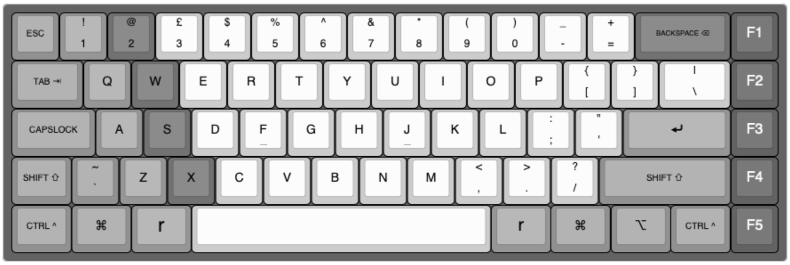
6.1 ASCII Mode

In ASCII mode, the keyboard internally manages most of the features you might expect from a keyboard – modifiers, caps lock, key repeat etc, and presents the resulting characters as standard ASCII over the UART interface.

When used with the rosco_m68k, the standard keyboard routines in the firmware expect the keyboard to be in ASCII mode by default.

In ASCII mode, your keyboard will "just work" like a keyboard – all common keyboard features (including modifier keys, caps lock / caps word, etc) are managed transparently by the keyboard.

For example, In this mode, pressing the 'A' key will send ASCII 0x61, which prints as a lower-case 'a'. Holding 'Shift' and pressing 'A' will send ASCII 0x41, which prints as upper-case 'A'. When caps-lock mode is enabled, this is reversed.


ESC ! 1 @ 2 £ 3 $ 4 % 5 ^ 6 & 7 * 8 (9) 0 _ - + = BACKSPACE ⌫ F1
TAB ⇥ Q W E R T Y U I O P { [}] | \ F2
CAPSLOCK A S D F G H J K L : ; " ' ↵ F3
SHIFT ⇧ ~ ` Z X C V B N M < , > . ? / SHIFT ⇧ F4
CTRL ^ ⌘ r r ⌘ ⌥ CTRL ^ F5

6.1.1 Command Keys

In ASCII mode, the command keys have no function.

6.1.2 Function Keys

In ASCII mode, the function keys have no function (since they do not have an equivalent in ASCII).

6.1.3 Caps Lock

Pressing the 'CAPS LOCK' key will lock the keyboard into capital-letter mode. The CAPS LOCK light will illuminate, and the mode will remain engaged until the 'CAPS LOCK' key is pressed again.

The behaviour is similar to if the 'SHIFT' key is held down, but it only modifies letters – numbers and symbols will be unshifted. To type lowercase letters when 'CAPS LOCK' is active, the 'SHIFT' key may be held.

6.1.4 Caps Word

If you want to type a single word in all-capitals, instead of using 'CAPS LOCK' mode, you can use 'CAPS WORD' mode instead.

This mode is engaged by pressing both 'SHIFT' keys at the same time. This will cause the 'CAPS LOCK' light to blink, indicating the mode is active.
When typing in 'CAPS WORD' mode, the behaviour is the same as for 'CAPS LOCK', except the mode will be automatically disengaged when any word-breaking character is typed.

In 'CAPS WORD' mode, you can press the 'CAPS LOCK' key at any time before a word-breaking character is typed to switch to regular 'CAPS LOCK' mode (cancelling the auto-disengagement of the mode).

To cancel 'CAPS WORD' mode, press both 'SHIFT' keys a second time, or type any word-breaking character.

6.2 Scancode Mode

In Scancode mode, the rosco_m68k keyboard simply sends make/break scancodes to the host indicating the specific keys that have been pressed and released.

In this mode, all keyboard features are controlled by the host driver software – caps lock, modifiers etc are simply scancodes – it is up to driver software to interpret these keys and maintain their state in software.

In this mode, the number of keys that can be pressed at once is unlimited, and all will be reported to the host (in scan order).

Mouse streaming can also be enabled in this mode, to allow fully interrupt-driven drivers.

Driver writers: See later sections in this manual for useful information on scancode formats and commands!

7. Pass-through mode

In advanced usage, you may wish to connect the keyboard to a device that only supports a single serial channel, and where that channel is also used for terminal output / uploading programs and so on.

Passthrough mode is designed for this purpose.

To use Passthrough mode, short JP3, and connect the keyboard to your computer as normal.

Then connect your usual terminal equipment (or USB-Serial converter) to J6. Once this connection is made, the keyboard will act as a transparent device, unless keys are being sent.

Obviously when transferring files etc using this feature, you should not press keys during the transfer as the keyboard is not disabled – and the keypresses will be inserted into the data stream which could cause the transfer to fail.

In pass-through mode, commands are disabled – any commands sent will be simply passed through without being processed by the keyboard.

Note that the TX/RX pins should be swapped when connecting to USB-Serial adapters, and that the pass-through serial will operate at the same bitrate as the main interface – either 9600 or 115200 depending on JP1.

8. Scan-code Format

In scan-code mode (activated via an appropriate command, or at startup by shorting JP2) the keyboard simply sends make/break codes for individual keys as the matrix scan progresses. In this mode, there are no limitations on which keys can be pressed and held at the same time.

```
 *     Down? | Row                    | Column
 *     ------|------------------------|--------------------------------
 *     1     | 1      | 1      | 1    | 1      | 1      | 1      | 1
 *     ------|------------------------|--------------------------------
 *     MSB   |                        |                        | LSB
 *
 * Row and column are 1-based in this scheme.
 * Rows six and seven represent special things:
 *
 *   PS/2 Mouse(*) - Packets always start with an "up" code for row 6, col 0
 *                   Four bytes follow (containing Intellimouse PS/2 packet)
 *
 *   SPI(*)     - Each byte received via the SPI interface will result in two
 *                scancode bytes, both identified as UP, column 7, with the low
 *                nibble containing half of the received byte, e.g. 0b0111nnnn.
```

(* - Selected models only)

9. Key Mappings in ASCII mode

No CAPS LOCK, Unshifted
```
    27,  '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', 8,   0,
    '\t','q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '[', ']', '\\',0,
    0,   'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ';', '\'',0,   '\r',0,
    0,   '`', 'z', 'x', 'c', 'v', 'b', 'n', 'm', ',', '.', '/', 0,   0,   0,
    0,   0,   0,   0,   0,   0,   ' ', 0,   0,   0,   0,   0,   0,   0,   0,
```

No CAPS LOCK, Shifted
```
    27,  '!', '@', '#', '$', '%', '^', '&', '*', '(', ')', '_', '+', 8,   0,
    '\t','Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', '{', '}', '|', 0,
    0,   'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', ':', '"', 0,   '\r',0,
    0,   '~', 'Z', 'X', 'C', 'V', 'B', 'N', 'M', '<', '>', '?', 0,   0,   0,
    0,   0,   0,   0,   0,   0,   ' ', 0,   0,   0,   0,   0,   0,   0,   0,
```

CAPS LOCK, Unshifted
```
    27,  '1', '2', '3', '4', '5', '6', '7', '8', '9', '0', '-', '=', 8,   0,
    '\t','Q', 'W', 'E', 'R', 'T', 'Y', 'U', 'I', 'O', 'P', '[', ']', '\\',0,
    0,   'A', 'S', 'D', 'F', 'G', 'H', 'J', 'K', 'L', ';', '\'',0,   '\r',0,
    0,   '`', 'Z', 'X', 'C', 'V', 'B', 'N', 'M', ',', '.', '/', 0,   0,   0,
    0,   0,   0,   0,   0,   0,   ' ', 0,   0,   0,   0,   0,   0,   0,   0,
```

CAPS LOCK, Shifted
```
    27,  '!', '@', '#', '$', '%', '^', '&', '*', '(', ')', '_', '+', 8,   0,
    '\t','q', 'w', 'e', 'r', 't', 'y', 'u', 'i', 'o', 'p', '{', '}', '|', 0,
    0,   'a', 's', 'd', 'f', 'g', 'h', 'j', 'k', 'l', ':', '"', 0,   '\r',0,
    0,   '~', 'z', 'x', 'c', 'v', 'b', 'n', 'm', '<', '>', '?', 0,   0,   0,
    0,   0,   0,   0,   0,   0,   ' ', 0,   0,   0,   0,   0,   0,   0,   0,
```

CONTROL (disregards shift / caps modifiers)
```
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   30,  0,   0,   0,   0,   31,  0,   0,   0,
    0,   0,   17,  23,  5,   18,  20,  25,  21,  9,   15,  16,  27,  29,  28,  0,
    0,   0,   1,   19,  4,   6,   7,   8,   10,  11,  12,  0,   0,   0,   '\r',0,
    0,   0,   0,   26, 24,   3,   22,  2,   14,  13,  0,   0,   127, 0,   0,   0,
    0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,   0,
```

9.1 Extended Characters

In order to provide basic support for certain non-ASCII characters, the OPTION key, in combination with other keys, allows some two- and three-byte UTF-8 characters to be typed - whether these are supported will depend on support in your retro / homebrew host computer!

These characters are typed by holding OPTION and using the number keys:

¡ € £ ¢ ∞ § ¶ ° ª º

The OPTION key also has another trick up its sleeve – holding OPTION turns the W, A, S and D keys into a regular “inverted-T” set of cursor keys.

10. Command Reference

Your rosco_m68k Keyboard supports a number of commands that software can send via the UART interface to control various aspects of the keyboard operation.

Using these commands, the mode can be changed, LEDs can be controlled, and advanced functions can be enabled or disabled.

10.1 Sending Commands

Commands are sent via the same UART interface used to receive characters or scancodes from the keyboard.

When the keyboard recognises a valid command, it will respond with ACK (0xff). An invalid command (or negative response) will result in NAK (0x00).

Where commands require an argument byte, the keyboard will acknowledge each byte (with either ACK or NAK as appropriate), to avoid the situation where an invalid command is sent, and then the argument is interpreted as a separate, unrelated command.

10.2 Commands

Hex	Name	Args	Description
0x1	LED_POWRED	1	Set intensity of power RED[0][1]
0x2	LED_POWGRN	1	Set intensity of power GREEN[0]
0x3	LED_POWBLU	1	Set intensity of power BLUE[0][1]
0x4	LED_CAPS	1	Enable / Disable CAPS LOCK LED[2]
0x5	LED_DISK	1	Enable / Disable DISK LED
0x6	LED_EXTRED	1	Set intensity of extension RED[3]
0x7	LED_EXTGRN	1	Set intensity of extension GREEN[3]
0x8	LED_EXTBLU	1	Set intensity of extension GREEN[3]

Hex	Name	Args	Description
0x10	MODE_SET	1	Set mode (0 = Scan, 1 = ASCII)
0x11	RPT_DELAY_SET	1	Set repeat delay
0x12	RPT_RATE_SET	1	Set repeat rate
0x20	MOUSE_DETECT	0	Detect a mouse (ACK if detected)
0x21	MOUSE_STRM_ON	0	Enable mouse reports[1][4]
0x22	MOUSE_STRM_OFF	0	Disable mouse reports[1]
0x23	MOUSE_REPORT	0	Request mouse status[1][5]
0x24	MOUSE_SET_RATE	1	Set mouse sample rate[1][6]
0x25	MOUSE_SET_RES	1	Set mouse resolution[1][6]
0x26	MOUSE_SET_SCALE	1	Set mouse scale[1][6]

Hex	Name	Args	Description
0x30	SPI_ENABLE	0	Enable SPI interface[1]
0x31	SPI_DISABLE	0	Disable SPI interface[1]
0xf0	IDENT	0	Request identification packet[7]
0xf1	RESET	0	Reset keyboard

10.2.1 Notes

[0] – LED intensity byte follows; some configurations may treat as a Boolean
[1] – Selected models only
[2] – In ASCII mode, the state of the LED may change based on user action
[3] – Depending on hardware configuration, may be individual LEDs
[4] – Will initialize mouse. NAK if no mouse detected. Requires JP4 open
[5] – If a mouse is connected, this will obtain a position report – see below
[6] – Valid values for sample rate, resolution and scale:

The argument to the MOUSE_SET_RATE command specifies the sample rate which should be selected.

Sample rate must be one of 10, 20, 40, 60, 80, 100, and 200 (which translates directly to samples per second).

Valid resolution values for MOUSE_SET_RES:

Value	**Resolution**
0x0	1 count per mm
0x1	2 counts per mm
0x2	4 counts per mm
0x3	8 counts per mm

Valid scaling values for MOUSE_SET_SCALE:

Value	**Resolution**
0x1	1:1 scaling
0x2	2:1 scaling

[7] – Identification packet format:

Byte(s)	**Content**
0x0-0x08	'rosco_kbd'
0x9	Mode (0 = scancode, 1 = ASCII)
0xa	Key count
0xb	LED count
0xc	Capability byte (see below)
0xd	Reserved
0xe	Reserved
0xf	ACK (0xff)

Where capability byte is bitwise OR of supported capabilities from:

```
CAP_KBD = 0x01
CAP_SPI = 0x02
CAP_I2C = 0x04
CAP_PWM = 0x08
CAP_PS2 = 0x10
```

The mouse position report contains a four-byte PS/2 Intellimouse® movement data packet, preceeded by a scancode indicating the start of a packet, and ACK indicating end of packet.

This looks like:

Byte(s)	Content
0x0	0x60 (Row 6, column 0)
0x1	PS/2 Status Byte
0x2	X position data
0x3	Y position data
0x4	Z position data (scroll wheel)
0x5	0x60 (ACK) **Only for packets resulting from a MOUSE_REPORT command**. For streaming reports, there will be no ACK and the packet will be 5 bytes only.

For details on the format of the status byte, see:

https://isdaman.com/alsos/hardware/mouse/ps2interface.htm

11. Example Code

This section contains some example C code showing how to interface directly with the keyboard and send commands / receive responses.

This code is written for the rosco_m68k firmware API but is easy to port to other systems.

11.1 Basic command interface

C Code

```c
#include <basicio.h>
#include <machine.h>

int        numdev;
CharDevice chardev[2];

void send_kbd(uint8_t byte)
{
    mcSendDevice((char)byte, &chardev[1]);
}

bool send_kbd_cmd(uint8_t cmd)
{
    send_kbd(cmd);
    uint16_t resp = mcInputchar() & 0xff;
    bool     ack  = (resp == 0xff);
    printf(" 0x%02x<-0x%02x=%s\n", cmd, resp, ack ? "ACK" : "NAK");

    return ack;
}

int kmain(void)
{
    numdev = mcGetDeviceCount();

    if (numdev < 2)
    {
        printf("No 2nd UART detected, exiting.\n\n");
        return;
    }

    for (int i = 0; i < 2; i++)
    {
        if (!mcGetDevice(i, &chardev[i]))
        {
            printf("failed to get device %d:", i);
            return 1;
        }
    }

    // use send_kbd_command here to send commands!
}
```

11.2 Sending commands & arguments

This example builds on the code above to allow sending commands with arguments to the keyboard.

C Code

```c
bool send_kbd_cmd_arg(uint8_t cmd, uint8_t arg)
{
    send_kbd(cmd);
    uint16_t resp  = mcInputchar() & 0xff;
    bool     ack   = (resp == 0xff);
    uint16_t resp2 = 0;
    bool     ack2  = false;
    if (ack)
    {
        send_kbd(arg);
        resp2 = mcInputchar() & 0xff;
        ack2  = (resp2 == 0xff);
        printf(" 0x%02x<-0x%02x=%s, 0x%02x<-0x%02x=%s\n",
               cmd,
               resp,
               ack ? "ACK" : "NAK",
               arg,
               resp2,
               ack2 ? "ACK" : "NAK");
    }
    else
    {
        printf(" 0x%02x<-0x%02x=%s <bad cmd?>\n", cmd, resp,
                ack ? "ACK" : "NAK");
    }
    return ack && ack2;
}
```

11.3 Display keyboard ID packet

Send the IDENT command and print out the resulting packet. The print_keycode routine here is great for debugging!

C Code

```
void print_keycode(int key)
{
    printf("0x%02x: ", key);
    if (key > 0x7f)
    {
        printf("high-bit+");
        key = key & 0x7f;
    }
    if (key < ' ')
    {
        printf("Ctrl-%c", key + 64);
    }
    else if (key == 0x7f)
    {
        printf("DEL");
    }
    else
    {
        printf("'%c'", key);
    }
}

int main(void)
{
    printf("\nrosco_keyboard ID: ");
    send_kbd(0xf0);

    uint8_t id_string[64] = {0};
    int     id_len        = 0;
    do
    {
        key                   = mcInputchar() & 0xff;
        id_string[id_len++] = key;
    } while (key != '~' && key != 0xff);

    printf("(ID length=%d bytes)\n", id_len);

    for (int i = 0; i < id_len; i++)
    {
        print_keycode(id_string[i]);
        printf("\n");
    }
}
```

11.4 Sending various commands

This brings the above examples together and sends various commands to the keyboard.

C Code

```c
    printf("\nASCII on    : ");
    send_kbd_cmd_arg(0x10, 0x01);

    printf("\nDisk LED on : ");
    send_kbd_cmd_arg(0x05, 0x01);

    printf("\nASCII echo test, press '~' for next test:\n");
    do
    {
        key = mcInputchar() & 0xff;

        print_keycode(key);
        printf("\n");
    } while (key != '~');

    printf("Disk LED off: ");
    send_kbd_cmd_arg(0x05, 0x00);

    printf("Mouse detect: ");
    bool mouse_detected = send_kbd_cmd(0x20);

    if (mouse_detected)
    {
        printf("Mouse was detected.\n");
    }
    else
    {
        printf("No mouse detected.\n");
    }
```

These examples should give a good starting point when writing software that interfaces with the keyboard.

By building on these routines you will be able to implement direct keyboard interfaces, game input code, drivers and other low-level code.

12. Care Instructions

Your rosco_m68k Keyboard has been carefully engineered to give you years of trouble-free operation. Follow the instructions in this section to ensure that your keyboard keeps working at its best.

Cleaning

Once you have built your keyboard PCB and finished soldering, we recommend removing excess flux using 99% isopropyl alcohol or flux-cleaning solution and employing a gentle scrubbing motion with a brush (an old toothbrush works great!).

Allow the PCB to dry completely before applying power.

WARNING

Isopropyl Alcohol is poisonous. Carefully follow the manufacturer's instructions when using it!

Dust & Dirt

Over time, you may find that dust and dirt build up on your keyboard.

If you have a bare PCB, we recommend giving it a quick blast with compressed or canned air every once in a while to clear this debris. If using a brush, use only a soft anti-static brush – static electricity may cause permanent damage.

Do not apply solvents or harsh cleaning chemicals to your PCB or case.

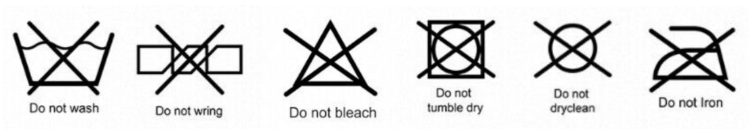
Do not immerse in water.

Key switches

You may choose to clean and lubricate your switches and stabilizers prior to assembly, and occasionally afterward.

We recommend a professional keyboard lubricant be used for this purpose. Carefully follow the manufacturer's instructions.

General Care



13. Credits

The rosco_m68k® Keyboard would not be possible without the help of these wonderful people from the community, to whom we are forever indebted.

- **@Malcolm Harrow**, **@0xTJ** and **@Xark** – For expert guidance on command interface design and infinite patience with testing and user manual revisions!
- **@Tankedup** – For tireless help and advice about making real things out of plastic, and for putting up with endless prototype 3D prints

Additionally, **massive thanks** to the whole community on Discord and elsewhere – there are too many of you to name, but without your support, encouragement and advice, there would be no rosco_m68k. Thank you all.

We'd also like to thank the wonderful people at Gateron in China for their patience with our questions about switches, and the engineers and support folks at JLCPCB and JLC3DP for incredible service and always catching our silly mistakes.

- https://www.gateron.com
- https://jlcpcb.com

Finally, and most importantly, we'd like to thank **you** – for believing in us, for buying the products, and supporting us to make more.

You are totally awesome – **thank you**!

14. Compliance Notices

All information contained in the product documentation (herein and online) and any additional information and documentation (including this notice) is correct as far as possible at the time of writing. Errors & omissions exempt.

To achieve compliance with local regulations regarding electro-magnetic interference (both transmission and receipt) the product may need to be operated in a suitable grounded enclosure with appropriate application-specific shielding. The Really Old-School Company Limited neither specify not supply such enclosures and recommend that expert guidance be sought where an enclosure is to be used.

The Really Old-School Company Limited does not authorize the use of any of its products in safety critical or life support applications where the failure or malfunction of the product can reasonably be expected to cause failure of the safety critical or life support system or to significantly affect its safety or effectiveness. This includes, but is not limited to, human life support, nuclear safety and control, air-traffic control, and vehicular control.

Products are not authorized for use in such applications under any circumstances.

All PCBs and components we supply are compliant with Restriction of Hazardous Substances in Electrical and Electronic Equipment (RoHS) regulations. Components that you source may not be compliant with the modern regulation, and compliance in finished kits you build will also depend on your choice of solder when building your board. Please dispose of any waste in accordance with relevant Waste Electrical and Electronic Equipment recycling (WEEE) regulations in your jurisdiction.