Mathkeeb™ Advanced User's Guide

2019-2020 edition

Table of Contents page number

Table of contents	1
Welcome/Getting Started	2
Reprogramming	3
Custom Firmware	5
Troubleshooting	10

Welcome!

Thank you for purchasing an Apogee Industries Mathkeeb! As you know, this keyboard was designed specifically to aid in efficiently typing mathematical expressions while writing equations, programming, or just for everyday use. However, we recognize that not all programs speak the same language! To solve this issue we have written up this small guide on how to customize the Mathkeeb to meet your personal needs, whatever they may be.

We're glad you've decided to take the next step and download the Advanced Users Kit. With this kit you will be able to load different hex files onto the keyboard's controller allowing it to speak different languages, or even have an entirely new layout. This guide will provide instructions on how to create and load your own custom firmware, or load one of the alternative firmwares which we have provided as a part of this kit.

Getting Started

In order to use this guide you will need the following resources:

1. An Apogee Industries Mathkeeb™
2. The included .zip folder

Note: Using some of the tools mentioned in this guide will require an internet connection.

Reprogramming

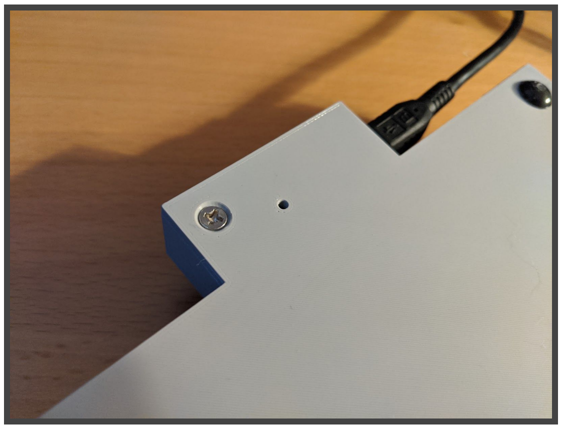
Reprogramming of your Mathkeeb is done by loading a new .hex file onto the Teensy Controller. This chapter will guide you through the process of doing so.

Loading .hex files onto your Mathkeeb is made simple via the use of the Teensy Loader software. This software can be downloaded from the PJRC website at the following link:

https://www.pjrc.com/teensy/loader.html

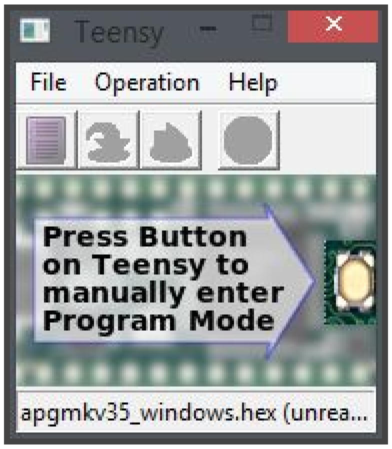
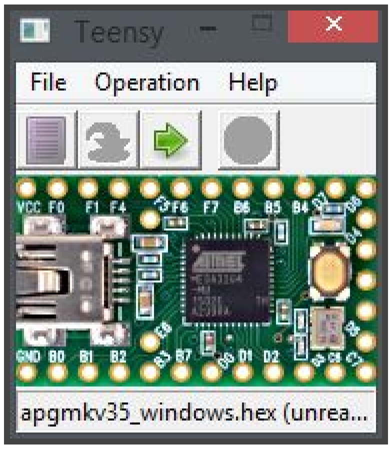
Once you have acquired the loader software, the loading process is as simple as pressing a few buttons.

The first button is located on the Mathkeeb itself. To access this button, flip the keyboard over and set it down on its keys. You should see a small pinhole near the usb port at the top (see below)



Please keep in mind there are delicate electronics inside this case! We recommend using a toothpick to push the button at the bottom of this hole. Once the button is pressed, you may hear a sound from your operating system indicating disconnection of a USB device. This confirms that you successfully hit the button.

Your Mathkeeb is now in programming mode. If you have the Teensy Loader software open, it should look something like this:


Teensy
File
Operation
Help
Press Button on Teensy to manually enter Program Mode
apgmkv35_windows.hex (unrea...


To use the loader, press the purple button on the left to open the .hex file you wish to load. Next, press the button in the middle with the green arrow pointing downwards to load the file to the keyboard. Finally, you can press the third button with the green arrow pointing to the right to reboot the keyboard. [**NOTE -** you will have to unplug the Mathkeeb and plug it back in to turn it back on]

You should now be able to use your mathkeeb with the newly loaded .hex file! If you are still having trouble loading a custom hex to the keyboard, please see the PJRC website for help.

https://www.pjrc.com/teensy/

Custom Firmware

We recognize that the firmware included with Mathkeeb and provided on our website may not cover the needs of all users. To remedy this, we have made available certain files that you can use, with a little work, to modify or create new firmware for Mathkeeb. In this guide, we discuss two methods to create new firmware .hex files for Mathkeeb. Which one you use will depend on your operating system, user needs, and your level of comfort with programming in C.

Method 1: Online Firmware Creator

This method involves using the Keyboard Firmware Builder by Ruiqi Mao, which can be found at https://kbfirmware.com/. We recommend using this method in the following cases:

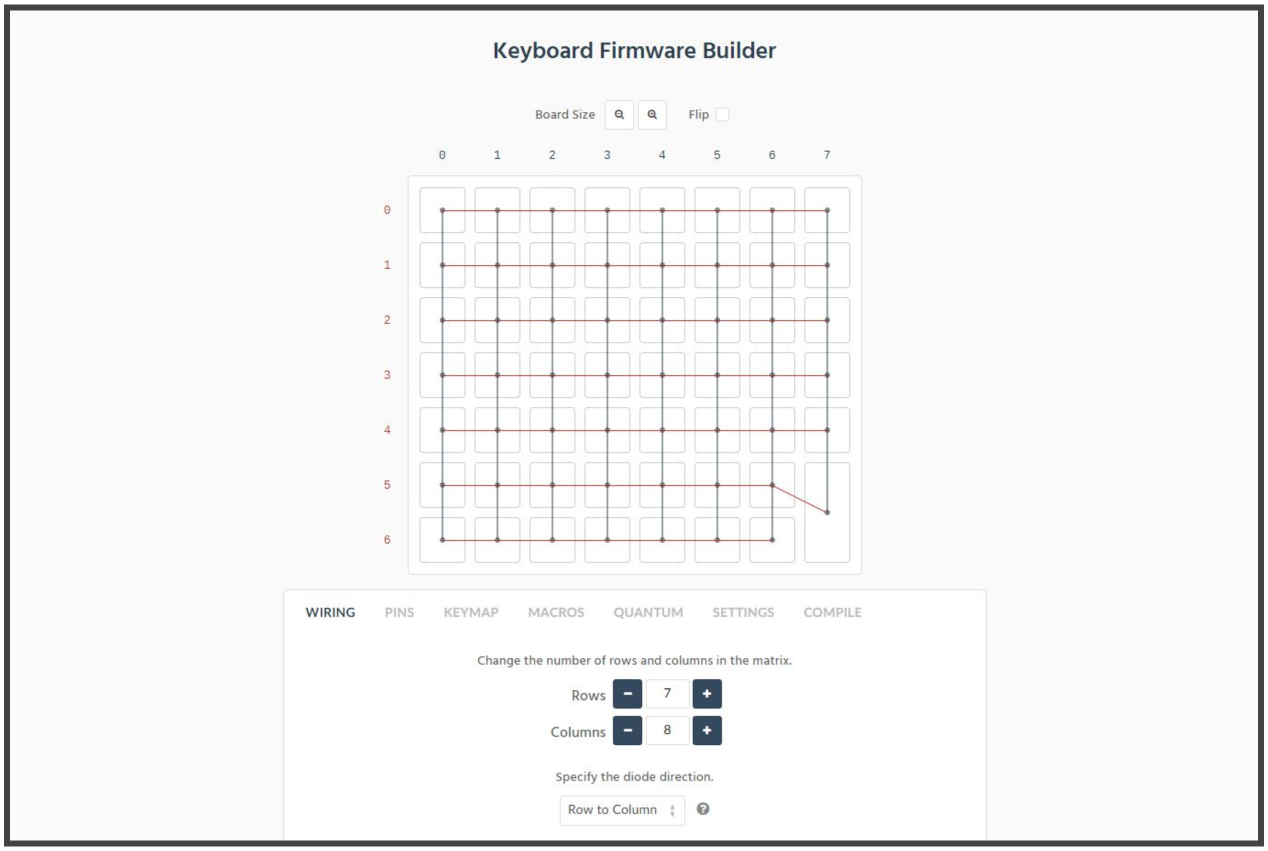
1. You **are** using the Windows version(s) of Mathkeeb's firmware
2. You are **not** using the Windows version(s), but **are** OK with losing functionality of any Unicode macro keys (on the Linux firmware, these are the keys which type π, Π, Σ, and ±)
3. You do not need any other features which are unavailable in the Keyboard Firmware Builder

Why these recommendations? The Linux/MacOS version of Mathkeeb's firmware types certain symbols using Unicode input. This option, while supported by the TMK firmware base, is not available on the Keyboard Firmware Builder site. The Windows version does not use Unicode input - instead, it types the symbol's Alt code, which is unique to the Windows operating system and does not require any special modifications in the source code.

In the online web tool, you can upload a specially formatted .json file containing the keyboard's layout, or start from scratch. To make modifying Mathkeeb's layout easier, we've included the necessary .json file in the .zip package for the Windows firmware versions, which can be downloaded on our website (on the same page you found this very guide).

NOTE: A .json is **NOT** included with the Linux/MacOS Mathkeeb firmware packages for the reason described above. Linux/MacOS users who wish to create new firmware using the online tool will need to download the Windows firmware package in order to get the .json file.

Upon uploading Mathkeeb's .json file to the Keyboard Firmware Builder site, you will see a screen like the image below:



Change the number of rows and columns in the matrix.

Rows 7

Columns 8

Specify the diode direction.

Row to Column

PLEASE DO NOT change any settings on the 'WIRING' or 'PINS' tabs unless you know what you are doing! This should never be necessary!

To edit the layout of your keyboard simply go to the 'KEYMAP' section and click which keys you wish to swap. Please note what the key is configured to do! If you are relocating a key that says 'macro' the new key must be set to use the same macro. Modifying macros can be done with ease under the 'MACROS' section.

When you are finished modifying the layout of your keyboard, modifying macros, etc, navigate to the 'SETTINGS' tab. Here you can rename your new layout and download a new .json file using the 'Save Configuration' button. **NOTE** - always save your .json file! It will be required if you wish to edit your current layout in the future!

After saving your layout, the new .hex file can be downloaded from the 'COMPILE' section. If you want to later add functionality not available on the Keyboard Firmware Builder site, you can download your layout as .zip, which will include the source code used to compile the .hex (see Method 2 for more information on this source code).

CONGRATULATIONS!! - You now have a firmware with your very own Mathkeeb layout! To see how to load this firmware onto your keyboard, please see Chapter 3 - Programming.

Method 2: Compiling a .hex from source

This method is a bit more involved, especially on Windows. We recommend compiling from source in the following cases:

1. You want to create new firmware retaining Unicode input or other functionality supported by TMK but not by the Keyboard Firmware Builder site
2. You are familiar with C and/or Makefiles, including the necessary dev tools
3. You are curious what the source code used to create Mathkeeb's firmware looks like

Before modifying any Mathkeeb keymap files you will need to set up a development environment. Linked below is the documentation website for TMK, the open-source firmware builder tool on which Mathkeeb and many other custom keyboards are based.

https://github.com/tmk/tmk_keyboard/blob/master/tmk_core/doc/build.md

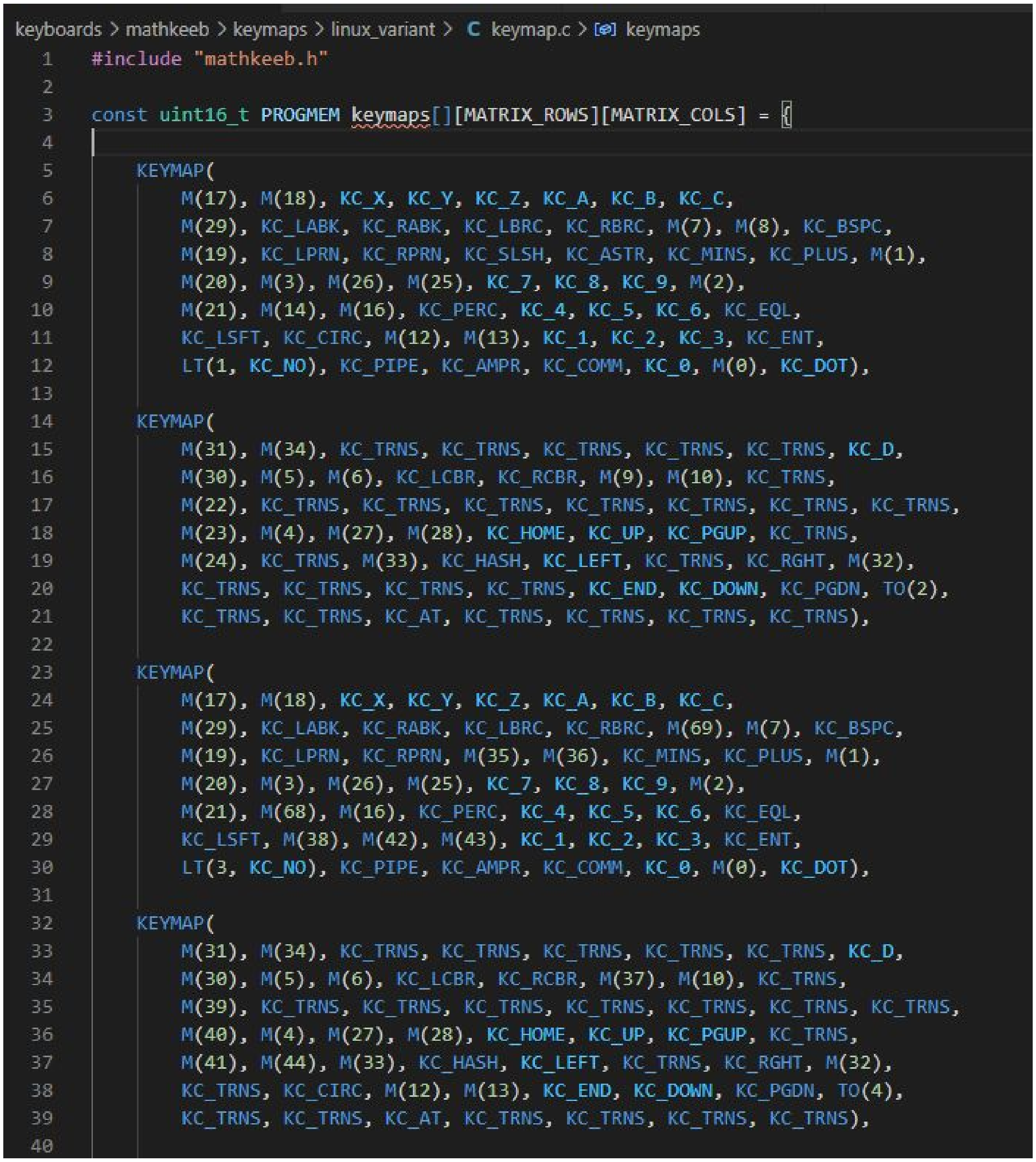
Click this link and follow Step 1: Install Tools. After doing this, you should have the necessary "toolchain" to compile a .hex for Mathkeeb. Note that the Mathkeeb does use the PJRC Teensy, and you will need the Teensy Loader to flash any new .hex files onto Mathkeeb (this is covered in the "Reprogramming" section above).

You do not need to bother with Step 2, as all of the source files for Mathkeeb are provided as a .zip on the advanced users page of our website. All you need to do is download and extract this .zip to a location of your choice.

NOTE: You should avoid working in a directory that has spaces in the file path (e.g. "Program Files") as this can cause problems. Stick to a path like C:\Users\Apogee\... instead.

For most changes that you might want to make to Mathkeeb's layout, you will need to open and edit one of the keymap.c files. Which one you choose will depend on your operating system: the keymap.c located in the 'windows' folder has macros for Alt codes, whereas the one in the 'linux' folder has macros for Unicode input.

First, open the keymap.c file in your text editor of choice. You should see something similar to the figure below. What you are looking at is each layer of Mathkeeb, expressed as a matrix of keycodes. You will notice that the default firmware only uses the first 8 layers, although TMK supports a maximum of 32 layers.



keyboards > mathkeeb > keymaps > linux_variant > keymap.c > keymaps

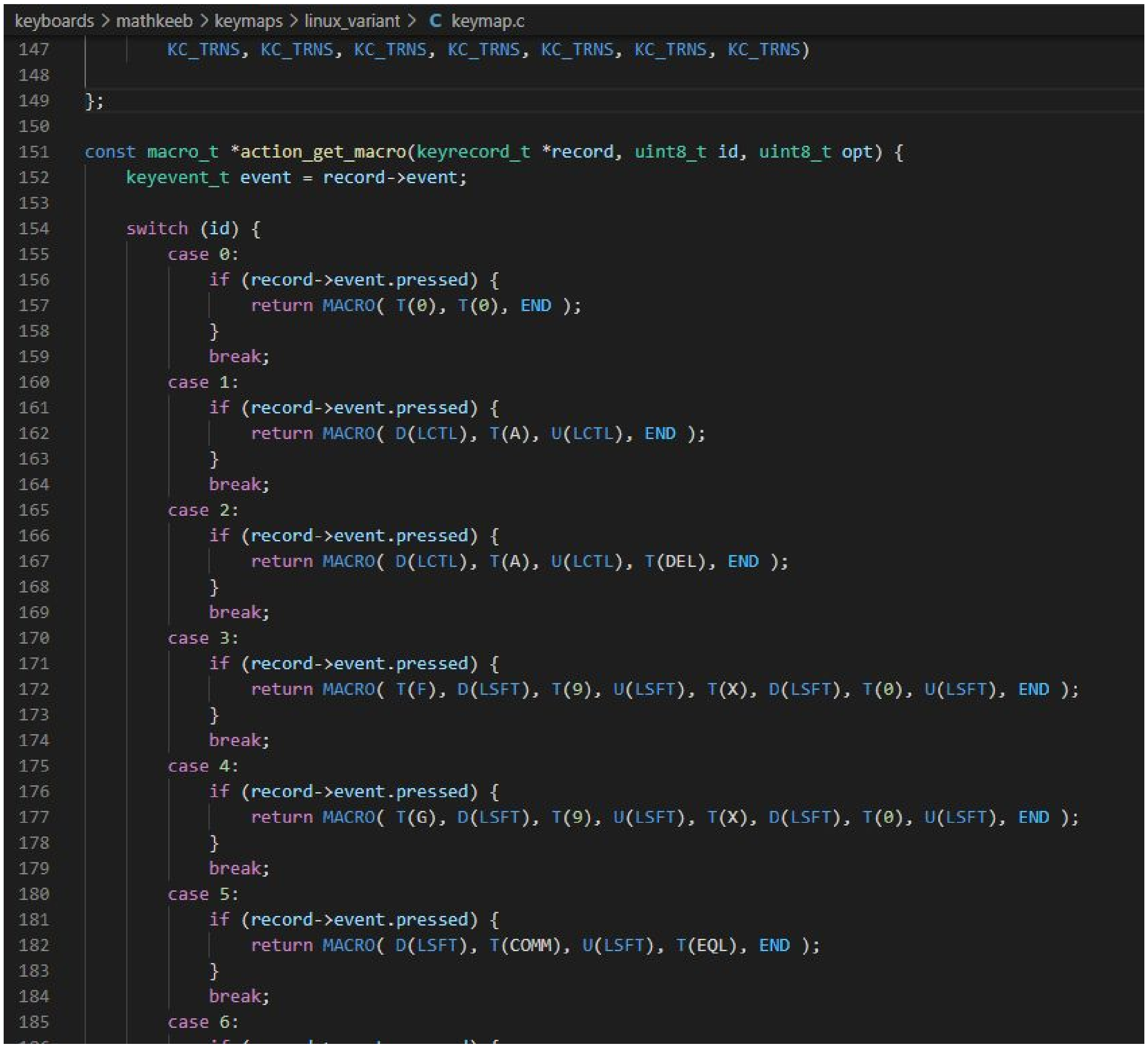
```c
#include "mathkeeb.h"

const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS] = {

    KEYMAP(
        M(17), M(18), KC_X, KC_Y, KC_Z, KC_A, KC_B, KC_C,
        M(29), KC_LABK, KC_RABK, KC_LBRC, KC_RBRC, M(7), M(8), KC_BSPC,
        M(19), KC_LPRN, KC_RPRN, KC_SLSH, KC_ASTR, KC_MINS, KC_PLUS, M(1),
        M(20), M(3), M(26), M(25), KC_7, KC_8, KC_9, M(2),
        M(21), M(14), M(16), KC_PERC, KC_4, KC_5, KC_6, KC_EQL,
        KC_LSFT, KC_CIRC, M(12), M(13), KC_1, KC_2, KC_3, KC_ENT,
        LT(1, KC_NO), KC_PIPE, KC_AMPR, KC_COMM, KC_0, M(0), KC_DOT),

    KEYMAP(
        M(31), M(34), KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_D,
        M(30), M(5), M(6), KC_LCBR, KC_RCBR, M(9), M(10), KC_TRNS,
        M(22), KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS,
        M(23), M(4), M(27), M(28), KC_HOME, KC_UP, KC_PGUP, KC_TRNS,
        M(24), KC_TRNS, M(33), KC_HASH, KC_LEFT, KC_TRNS, KC_RGHT, M(32),
        KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_END, KC_DOWN, KC_PGDN, TO(2),
        KC_TRNS, KC_TRNS, KC_AT, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS),

    KEYMAP(
        M(17), M(18), KC_X, KC_Y, KC_Z, KC_A, KC_B, KC_C,
        M(29), KC_LABK, KC_RABK, KC_LBRC, KC_RBRC, M(69), M(7), KC_BSPC,
        M(19), KC_LPRN, KC_RPRN, M(35), M(36), KC_MINS, KC_PLUS, M(1),
        M(20), M(3), M(26), M(25), KC_7, KC_8, KC_9, M(2),
        M(21), M(68), M(16), KC_PERC, KC_4, KC_5, KC_6, KC_EQL,
        KC_LSFT, M(38), M(42), M(43), KC_1, KC_2, KC_3, KC_ENT,
        LT(3, KC_NO), KC_PIPE, KC_AMPR, KC_COMM, KC_0, M(0), KC_DOT),

    KEYMAP(
        M(31), M(34), KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_D,
        M(30), M(5), M(6), KC_LCBR, KC_RCBR, M(37), M(10), KC_TRNS,
        M(39), KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS,
        M(40), M(4), M(27), M(28), KC_HOME, KC_UP, KC_PGUP, KC_TRNS,
        M(41), M(44), M(33), KC_HASH, KC_LEFT, KC_TRNS, KC_RGHT, M(32),
        KC_TRNS, KC_CIRC, M(12), M(13), KC_END, KC_DOWN, KC_PGDN, TO(4),
        KC_TRNS, KC_TRNS, KC_AT, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS),

```

Screenshot of one of Mathkeeb's keymap.c files, using VSCode

There are different types of keycodes:

- Ones beginning with "KC_" are letters or symbols that would appear on a normal keyboard
    - KC_TRANS indicates that the key is 'transparent', and will not do or type anything when pressed

- Those using “M( )” are macros, which are identified by a number and defined in the keymap.c file below the layer matrices
- Other functions include “LT( )” which toggles a particular layer when held, and “TO( )” which switches to a particular layer
- A more complete list of valid keycodes and functions can be found here: https://github.com/tmk/tmk_keyboard/blob/master/tmk_core/doc/keymap.md


keyboards > mathkeeb > keymaps > linux_variant > keymap.c

```c
        KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS, KC_TRNS)

};

const macro_t *action_get_macro(keyrecord_t *record, uint8_t id, uint8_t opt) {
    keyevent_t event = record->event;

    switch (id) {
        case 0:
            if (record->event.pressed) {
                return MACRO( T(0), T(0), END );
            }
            break;
        case 1:
            if (record->event.pressed) {
                return MACRO( D(LCTL), T(A), U(LCTL), END );
            }
            break;
        case 2:
            if (record->event.pressed) {
                return MACRO( D(LCTL), T(A), U(LCTL), T(DEL), END );
            }
            break;
        case 3:
            if (record->event.pressed) {
                return MACRO( T(F), D(LSFT), T(9), U(LSFT), T(X), D(LSFT), T(0), U(LSFT), END );
            }
            break;
        case 4:
            if (record->event.pressed) {
                return MACRO( T(G), D(LSFT), T(9), U(LSFT), T(X), D(LSFT), T(0), U(LSFT), END );
            }
            break;
        case 5:
            if (record->event.pressed) {
                return MACRO( D(LSFT), T(COMM), U(LSFT), T(EQL), END );
            }
            break;
        case 6:
```

The same keymap.c file, showing some of the macro definitions

Macros are defined in the switch statement after the layer matrices. Inside the “MACRO( )” call, provide a comma-separated list of the actions you want to be executed by the macro, in the order you want them to be executed. Defining macro actions works a little differently from how the layer matrices were defined above:

- “T( )” indicates a key which is typed once

- "D( )" indicates a key which is held down
- "U( )" indicates to release a held-down key (used in combination with "D( )")
- "END" indicates the end of the macro - each macro definition must have END as the last item
- The argument of the T, D, and U actions above is a keycode
    - Note that you do not include the "KC_", only the letter/key itself
    - If in doubt, look at the 80 macros which are defined already
- The list of valid macro functions can be found here (scroll down a bit): https://github.com/tmk/tmk_keyboard/blob/master/tmk_core/doc/keymap.md

If you are confused by the macro definitions, note that the first macro (case 0) types 00, the 2nd macro (case 1) does Ctrl+A, the 4th macro (case 3) types f(x), etc. We are not really sure if there is a maximum number of macros allowed, so if you would like to test that and make 100+ macros please be our guest and let us know how it goes.

When satisfied with your new firmware, simply follow the process outlined in the "Build Firmware" section of the TMK docs linked above. You will need to navigate to the directory containing the directo(ies) containing the keymap.c file and then run the make command as shown.

If you do not specify a variant, you will get files that look like "mathkeeb_windows.hex" or "mathkeeb_my_new_version.hex" where "my_new_version" is the name of the directory containing the keymap.c file you created.

NOTE: In Windows, you will need to run the make command from the GNU-like terminal (MinGW-64 or MSYS2) you downloaded in step 1, it does NOT work from Windows Command Prompt or Powershell.

Troubleshooting

Q. Mathkeeb doesn't type anything!

A. Make sure you aren't in programming mode, and have properly rebooted Mathkeeb with the correct firmware. Unplug and re-plug the keyboard as well. If it still doesn't work, revert to the default firmware. If that doesn't work, please refer to the troubleshooting section of the Assembly Guide.

Q. [Key] doesn't type what I want it to!

A. Double check your matrices, and make sure you still have the 8x7 grid, with everything in the right place. Also check that you are on the correct layer, and that your layer toggle functions are on the right keys.

Q. [Key] is acting like it's held down, even though I only pressed it once!

A. Double check your macros. You may have accidentally used the hold-down function insead of the type-once function (it happened to us in testing!). Also make sure all of your D( ) functions are paired with a U( ) function.

Q. I see a bunch of red squiggles in my IDE when I open Mathkeeb's keymap.c files!

A. Surprisingly, this was normal for us as well. We are not really sure what causes it, but to check if you actually have errors, try compiling a hex using the Makefile method outlined above. If your keymap fails to compile, please refer to documentation or troubleshooting on the TMK website.