

QMK firmware running on Xwhatsit

Table of contents

Table of contents	1
Licensing	2
Glossary	2
Advantages / Disadvantages	3
Advantages of QMK firmware running on Xwhatsit	3
Advantages of original Xwhatsit firmware	3
Supported Keyboards	4
Keyboard naming convention	4
Entering bootloader mode	5
The hardware method to enter bootloader mode	5
Putting the xwhatsit model F controller into bootloader mode	5
Putting the xwhatsit beamspring rev. 4 controller into bootloader mode	6
Putting the wcass controller into bootloader mode	7
Putting one of the universal controllers into bootloader mode	8
If you have a Pro Micro-like board with “old” Caterina bootloader	8
If you have a Pro Micro-like board with “new” Caterina bootloader	10
Trap for young players when using Caterina based bootloaders on Linux	10
Trap for young players using 5V Pro Micros bought from ebay	10
How do I tell if I have a 5V Pro Micro or a 3.3V Pro Micro?	11
If you have a Pro Micro-like board with other type of bootloader	11
Is it possible to use a 3.3V Pro Micro with TH controller?	12
Proton C board is not currently supported.	12
Entering bootloader mode using the host-side utility	12
Entering bootloader mode using the QMK “RESET” keycode	13
Entering bootloader mode using MAGIC_KEY_BOOTLOADER	14
Entering bootloader using “QMK Bootmagic”	14
Getting/Building the host-side utility	14
Building the firmware	16
Keymap rules	16
Where to build	16
Building the firmware using configurator	16
Building the firmware locally on the command line	17
Before flashing, if switching from original xwhatsit firmware	18
Flashing	18

Common usage tips	19
Debugging	19
Erasing EEPROM	19
How Capacitive Sensing Works	20
How Auto-Calibration works	20
Keypress Monitor	20
Signal Level Monitor	20
Debugging freshly-built controllers when it just doesn't work	20
Reverting back to original xwhatsit firmware	21
Testing newly-built controllers, without keyboards	22
Using expansion headers	24
Solenoids & solenoid drivers	24
Lock lights	26
Lock light pins available on all controllers	26
If solenoid is not in use	26
Additional lock light pins available on the "standard" SMD Model F controllers:	27
Additional lock light pins available on the "kishsaver" SMD Model F controllers:	27

Licensing

This document is dual-licensed under:

- Attribution-ShareAlike 4.0 International (CC BY-SA 4.0)
- GNU GPL v2

Glossary

- **QMK Firmware** is an open source firmware used by many keyboards:
<https://qmk.fm/>
- **Xwhatsit** is a type of controller PCB designed to connect to IBM keyboards that employ capacitive sensing. The name comes from Geekhack user "xwhatsit" who first implemented it.
 - <https://static.wongcornall.com/ibm-capsense-usb-web/ibm-capsense-usb.html>
- **Original Xwhatsit firmware** is an open source firmware which was released by Geekhack user "xwhatsit" together with his controller designs.
- **QMK firmware running on Xwhatsit-type controllers** is an independently implemented firmware by DT user "pandrew", that runs on Xwhatsit's controller PCBs (and some newly developed xwhatsit-like controllers), using QMK as a base. This document is about this firmware.

Advantages / Disadvantages

This is an incomplete list. Many of these advantages or disadvantages could be equalized if someone takes on the task to improve one or the other firmware.

Advantages of QMK firmware running on Xwhatsit

- Better auto-calibration
 - It uses multiple thresholds (threshold bins), and sorts keys in these threshold bins. Original xwhatsit firmware only uses a single threshold for all keys, which may not be perfectly adequate for all keys.
 - Auto-calibration doesn't require configuring calibration pads, and it is much more reliable than original xwhatsit firmwares auto-calibration
- With QMK it is easier to understand how to assign an action to a key. QMK firmware already knows about the layouts of all the keyboards it supports. So to configure your keymap, you can just drag-and-drop in QMK configurator the action to the key. On original Xwhatsit firmware you have to press the key to figure out in which matrix position it is, and then assign an action to that matrix position.
- QMK firmware running on Xwhatsit has debugging features that allow the user to more easily figure out what is wrong.
- There are many non-xwhatsit-specific features in QMK that are not present in original Xwhatsit firmware, for example: more layers, more complex macros, tap dance, more debouncing methods, ...
- Supports the new Universal Through-Hole controller. (Original XWhatsit could also support this controller, but at the moment it's not implemented)

Advantages of original Xwhatsit firmware

- Original xwhatsit firmware does not require the firmware to be rebuilt to change the keymap. QMK firmware is designed to be rebuilt and reflashed after any keymap change.
 - There is something in QMK called VIA which apparently allows a number of dynamically editable layers, but this is not currently supported by QMK firmware running on Xwhatsit controllers.
- It is easier to support a completely new keyboard with the original xwhatsit firmware, and no programming knowledge is required. To add a new keyboard to QMK firmware, there are a number of steps required, and some programming knowledge might be necessary.
- Maybe (the writer has not explored these yet, and anything written below maybe incorrect:)
 - NKRO may be implemented in a way that better supports older bioses in xwhatsit. In QMK it may be necessary to switch on/off NKRO with a special keycode for access to bioses.

*NOTE: there have been a couple improvements to original xwhatsit firmware since Tom Wong Cornall's latest release, namely: Better Debouncing by joc, and corrected handling of

momentary function keys by pandrew. Both of these changes can be found here:
https://github.com/purdeaandrei/ibm_capsense_usb_mods

Supported Keyboards

Keyboard naming convention

All keyboards currently follow the following naming convention:
xwhatsit/\$(VENDOR)/\$(KEYBOARD)/\$(CONTROLLER)

\$(VENDOR) is one of:

- **brand_new_model_f** - Ellipse's brand new model F reproductions
 - Official site: <https://www.modelfkeyboards.com/>
 - Deskthority discussion thread: <https://deskthority.net/viewtopic.php?t=11046>
 - NOTE: brand new model f keyboards come with the **wcass** controller
- **ibm** - keyboards originally made by IBM, or that sport the IBM trademark, including beamspring and model F keyboards
- **sneakyrobb** - Sneakyrobb's new beamspring keyboards:
 - Deskthority discussion thread: <https://deskthority.net/viewtopic.php?t=21851>

\$(KEYBOARD) is the name of the keyboard, please be aware that some keyboards exist from multiple vendors, but the sense PCB is incompatible, so for example, these f62 keyboards are different, and you have to select the correct one:

- xwhatsit/brand_new_model_f/f62/wcass
- xwhatsit/ibm/f62/wcass

\$(CONTROLLER) is the capsense controller PCB used in the keyboards, and it's one of:

- **xwhatsit** - The original model F rev. 1 or rev. 2 controller, as released by geek hack user xwhatsit, aka Tom Wong-Cornall.
 - Official site:
<https://static.wongcornall.com/ibm-capsense-usb-web/ibm-capsense-usb.html>
- **xwhatsit_rev4** - The original beamspring rev. 4 controller, as released by geek hack user xwhatsit, aka Tom Wong-Cornall.
 - Note: rev 1/2/3 controllers are not supported at the moment
- **wcass** - The smaller xwhatsit model F PCB that was developed by DT user "wcass"
 - Deskthority discussion thread:
<https://deskthority.net/viewtopic.php?t=13479>
- **universal** - This firmware variant supports multiple controllers:
 - The through hole universal model F and Beamspring controller, developed by DT users kmonv2017 and listofoptions:
 - Deskthority discussion thread:
<https://deskthority.net/viewtopic.php?t=23406>
 - Compact Beamspring Controller
 - <https://deskthority.net/viewtopic.php?f=50&t=24512>
 - SMD Model F controller ("standard", and 4704 kishsaver-class too)

- <https://deskthority.net/viewtopic.php?f=7&t=24597>
- This used to be called “through_hole”, and has been renamed to “universal”, because it now supports multiple controllers.

For keyboards that come from the vendor with a supported controller, it will also be possible to use the naming “xwhatsit\$(VENDOR)\$(KEYBOARD)” directly, whereby the controller the keyboard comes with is selected automatically.

Ellipse’s brand new model F keyboards come by default with the “wcass” controller, but other controllers are also available for this keyboard, in case the user decides to switch it out.

For IBM there will be no default controller selected. The user has to know which controller they are using.

Entering bootloader mode

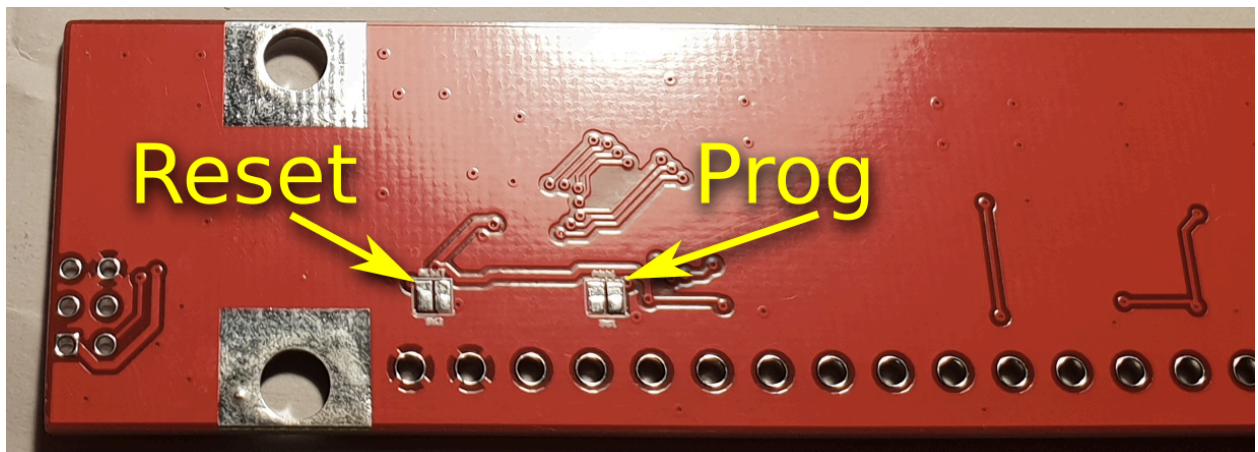
Before we start talking about how to flash QMK firmware, the user must be comfortable with the various ways entering bootloader mode.

The hardware method to enter bootloader mode

The hardware method to enter bootloader mode is always expected to work, even if you flash a wrong firmware to the controller. Depending on bootloader, it usually involves shorting two contacts while plugging the keyboard in, and releasing after. Sometimes it involves shorting two contacts immediately after the keyboard is plugged in. This should be used rarely, most of the time only to correct a mistake, because it involves opening up the keyboard case.

Putting the xwhatsit model F controller into bootloader mode

The original xwhatsit rev. 1 and rev. 2 model F controller has two pairs of shortable pads opposite side of the main CPU, as seen in the image below.



This controller is running an atmega32u2 chip, and usually has an atmel-dfu bootloader, and there are two ways to put it into bootloader mode:

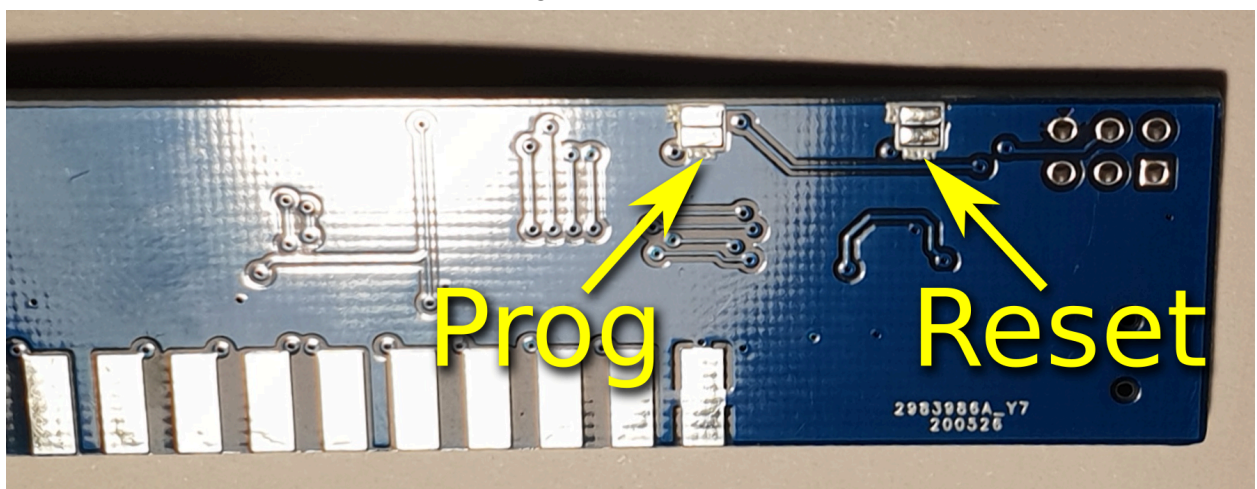
- Method 1:
 - Unplug keyboard
 - Short the two sides of “Prog”, keep shorting
 - Plug in keyboard
 - Release “Prog”
- Method 2:
 - Plug in keyboard
 - Short the two sides of “Reset”, keep shorting
 - Short the two sides of “Prog”, keep shorting
 - Release “Reset”
 - Release “Prog”

Method 1 will not allow the programmed firmware to run first before entering bootloader mode, so it is safer if the keyboard is generating invalid keystrokes.

The above methods only work, if the bootloader has not been replaced.

Putting the xwhatsit beamspring rev. 4 controller into bootloader mode

The original xwhatsit rev. 4 beamspring controller has two pairs of shortable pads opposite side of the main CPU, as seen in the image below.



This controller is running an atmega32u2 chip, and usually has an atmel-dfu bootloader, and there are two ways to put it into bootloader mode:

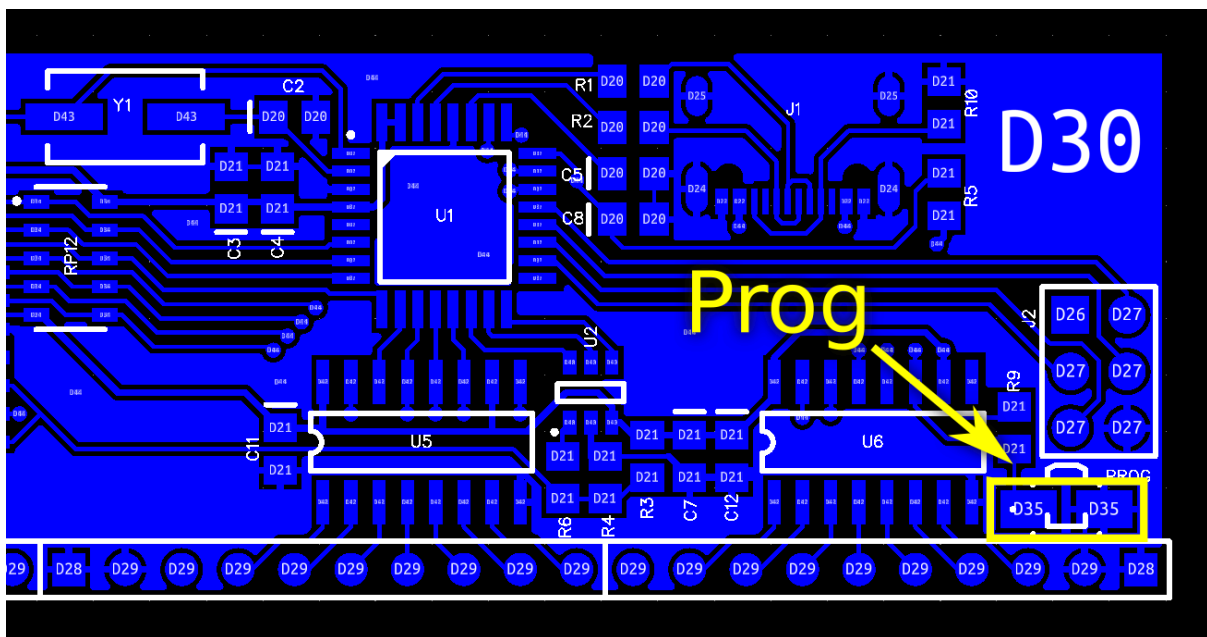
- Method 1:
 - Unplug keyboard
 - Short the two sides of “Prog”, keep shorting
 - Plug in keyboard
 - Release “Prog”
- Method 2:
 - Plug in keyboard
 - Short the two sides of “Reset”, keep shorting
 - Short the two sides of “Prog”, keep shorting
 - Release “Reset”
 - Release “Prog”

The above methods only work, if the bootloader has not been replaced.

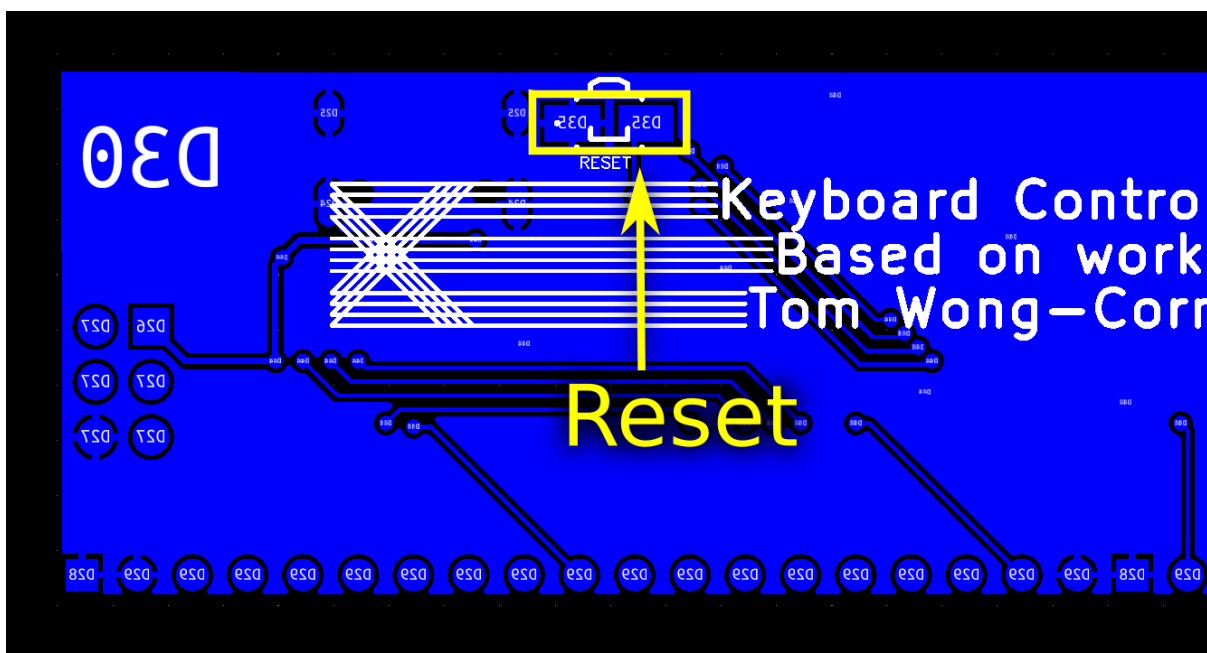
Putting the wcass controller into bootloader mode

The wcass controller has a number of variants that are similar, there are USB-C, microUSB and pin header variants for the USB connection. Other than the USB connector, all the variants are all expected to behave the same. Only the USB-C variant is pictured below. This controller has the Prog and Reset pairs of pads on opposite sides. This controller has larger Prog/Reset pads which could allow soldering small SMD switches to them, to make shorting those pads easier, however most people who built these boards chose not to mount any switches in these positions.

On the top side of the wcass PCB the Prog pads are in the corner, next to the solenoid/expansion header:



On the bottom side of the wcass PCB the Reset pads are just opposite to the microcontroller:



This controller is running an atmega32u2 chip, and usually has an atmel-dfu bootloader, and there are two ways to put it into bootloader mode:

- Method 1:
 - Unplug keyboard
 - Short the two sides of “Prog”, keep shorting...
 - Plug in keyboard
 - Release “Prog”
- Method 2 - only applicable if microswitches are soldered to the pads, otherwise it would be really hard to apply this method shorting the pads with a screwdriver:
 - Plug in keyboard
 - Press the Reset button
 - Press the Prog button
 - Release “Reset”
 - Release “Prog”

The above methods only work, if the bootloader has not been replaced.

When shorting the pads, make sure that you use something that makes good contact, maybe something that digs a little into the metal, to remove the surface oxidation layer on the pads. Using a simple screwdriver might take a few tries. Using sharp tweezers and pushing them a little into the pad so it leaves a mark, might work better.

Putting one of the universal controllers into bootloader mode

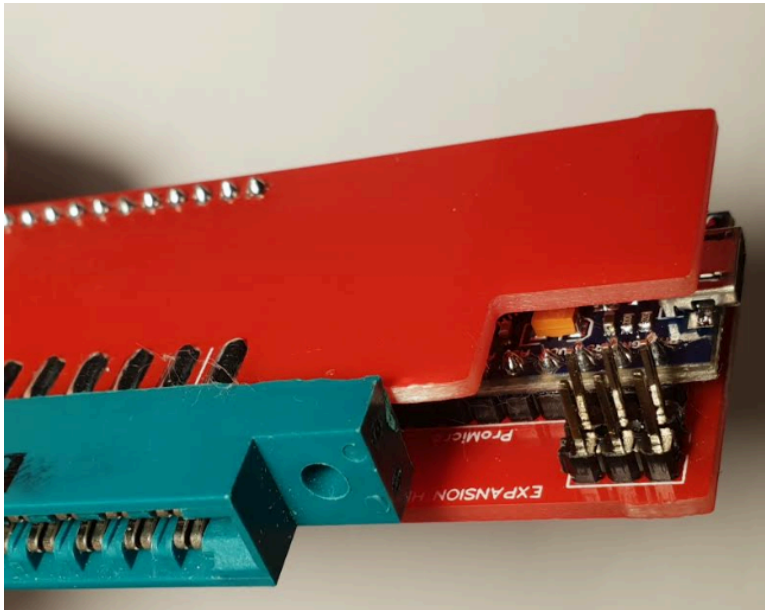
- 1) If you have a Pro Micro-like board with “old” Caterina bootloader

Most Pro Micro boards from ebay fall into this category.

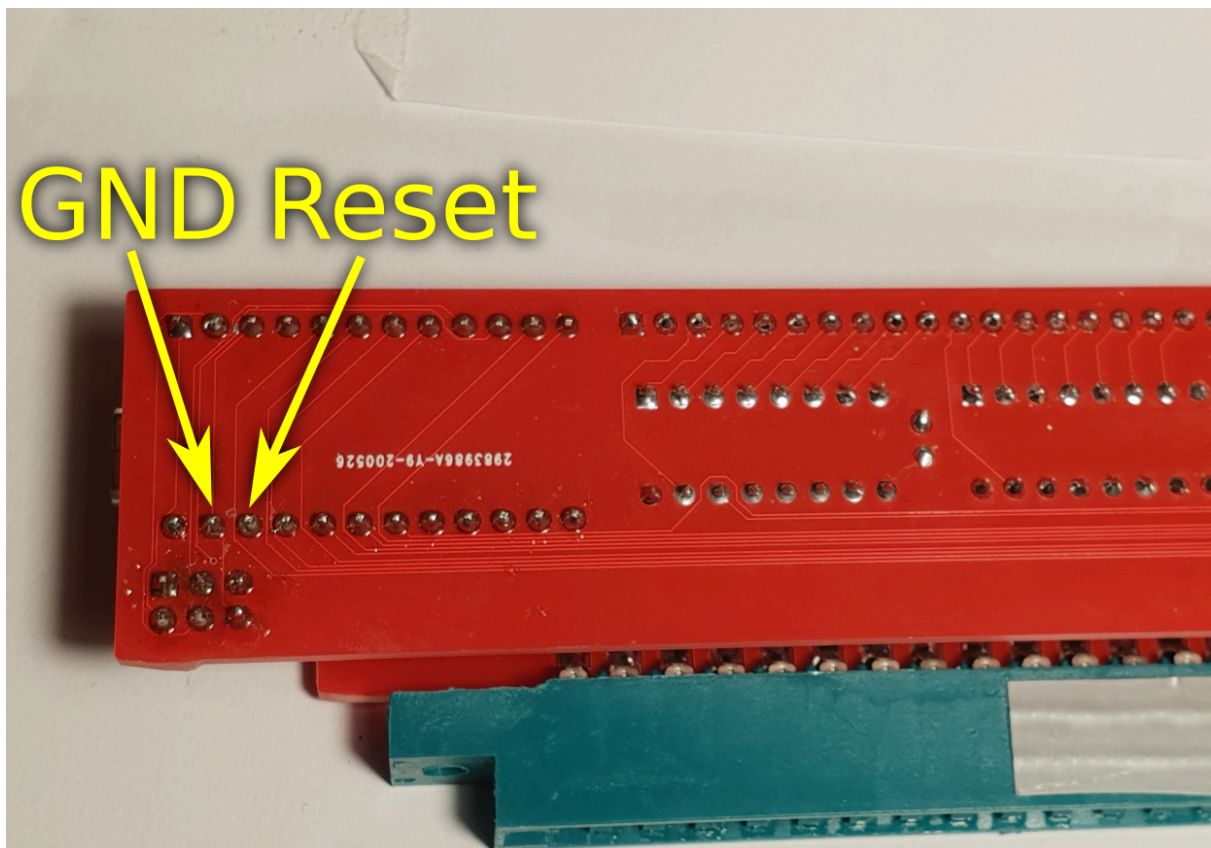
Pro Micros usually don't have PROG pads like the previously listed controllers. They usually run the Caterina bootloader, which uses the Reset pin to get the device into bootloader mode. As long as you don't have a plugboard covering the promicro, you have access to the relevant pins, marked with GND and RST:



However if a plugboard is mounted, the pins might not be easily accessible on this side:



However the pins will remain accessible on the bottom (but they are not marked, so you have to remember which ones they are). Often the bottom is more readily available especially if the board is connected to a beamspring keyboard:



- Method 1 to enter bootloader:
 - Plug in keyboard
 - Short and release the GND and Reset pins
 - There is an 8 second time interval during which the device is in bootloader mode, after that the normal firmware is started.
- Method 2 - if the keyboard is generating a large number of invalid keystrokes, and you can't use method 1, then this method will avoid starting the firmware the first time the keyboard is plugged in:

- Keyboard Unplugged
- Short the GND and Reset pins, keep them shorted
- Plug in keyboard
- Release the Reset
- There is an 8 second time interval during which the device is in bootloader mode, after that the normal firmware is started.

2) If you have a Pro Micro-like board with “new” Caterina bootloader

The “new” Caterina bootloader changes the behavior of the reset pin. Now a single reset will not cause the device to enter bootloader mode, instead the device must be reset twice in quick succession.

Please see the following document describing the behavior:

<https://cdn.sparkfun.com/datasheets/Dev/Arduino/Boards/32U4Note.pdf>

Some pro-micro compatible products, such as the “Sparkfun Qwiic Pro Micro” will also have a reset button, and you can also use that instead of shorting RST to GND.

3) Trap for young players when using Caterina based bootloaders on Linux

On linux the caterina bootloader registers as a /dev/ttyACM* device.

On some linux distributions ModemManager will grab any ttyACM device and tries to talk to it as soon as you plug in the Pro Micro. If ModemManager is allowed to do this, then flashing will either be impossible, or it will be very unreliable.

Possible Workarounds:

- Temporarily disable ModemManager, with something like “sudo systemctl stop ModemManager” or equivalent command. This will last until the next reboot.
- If you’re not using it, you can permanently disable ModemManager, with something like “sudo systemctl disable ModemManager” or equivalent command, or just uninstall the modemmanager package
- Tell ModemManager to not touch the Pro Micro
 - Add this line to an udev config:
 - ATTRS{idVendor}=="2341", ENV{ID_MM_DEVICE_IGNORE}="1"
 - The ModemManager documentation says: “This tag was originally applicable to TTY ports and only when running in DEFAULT or PARANOID filter policy types. Since 1.12, this tag applies to all filter types (including STRICT), and to all port types (not only TTYs), and is associated to the MM_FILTER_RULE_EXPLICIT_BLACKLIST rule.”
 - So you might need to also make sure that ModemManager is not in strict mode, for example on debian, edit /lib/systemd/system/ModemManager.service , look for the ExecStart= line, and edit the --filter-policy option and set it to default/paranoid/whitelist-only

4) Trap for young players using 5V Pro Micros bought from ebay

5V MicroUSB Pro Micros are supposed to be shipped with the “J1” jumper shorted. (J1 is next to the microusb port). However most ebay sellers sell Pro Micros with J1 not shorted from the factory. This is incorrect, but it will work somewhat. The result of not shorting J1 is that the VCC voltage will be lower because of the voltage drop on the 5V voltage regulator. The input voltage is low enough that the voltage regulator will not regulate anything, so all it does is drop some voltage for no good reason.

It is recommended to short the J1 jumper on all 5V Pro Micros.

The only exception would be split QMK keyboards, (in which case J1 should be left open, and RAW and VCC should be shorted), but that exception does not apply here.



5) How do I tell if I have a 5V Pro Micro or a 3.3V Pro Micro?

There are two methods:

a) Read the Crystal Method:

All 5V Pro Micros have 16MHz Crystals

All 3.3V Pro Micros have 8MHz Crystals

So read the writing on this component:



b) Measure VCC to GND method (only works if J1 is not currently shorted)

On some pro micro variants the crystal is small or it's not possible to read the frequency off of it. In that case you have to measure the voltage:

If the Pro Micro is of the 5V variant, then when you measure the voltage between VCC and GND, you will get a value higher than 4V. If you have a 3.3V pro micro then you should measure a nice regulated 3.3V voltage. This only works if J1 is not shorted.

c) You cannot rely on looking at the state of the J1 jumper to determine the type of Pro Micro you have.

6) If you have a Pro Micro-like board with other type of bootloader

In this case the instructions might differ.

For example the Elite-C Pro Micro comes with a type of DFU bootloader.

Please note that this has not been tested, and if you use one of these boards, you might need to change the `BOOTLOADER=` setting in `rules.mk` for non-hardware bootloader entry to work correctly.

7) Is it possible to use a 3.3V Pro Micro with TH controller?

Currently only 5V 16MHz Pro Micros are supported. 3.3V pro micros might work (although they have not been tested), if you do the following obligatory changes:

- `rules.mk`: `F_CPU = 8000000`
- `rules.mk`: `F_USB = $(F_CPU)`
- `config.h`: `CAPSENSE_HARDCODED_SAMPLE_TIME` -- divide by two and round up.
- NOTE: You will certainly observe lower performance

The above changes will still have the 3.3V Pro Micro run at 3.3V, which means that all other chips on the controller will also run at 3.3V. I have not done a detailed review if all other chips support this or not. It might work or it might not, however you can also run your 3.3V pro micro in 5V mode, if you short J1. The microcontroller chip will happily run at a higher voltage, but the slower crystal will still limit performance. So a probably needed change is to:

- Short J1.

NOTE: if you flash a 5V Pro Micro firmware to a 3.3V Pro Micro, it will most likely not even enumerate.

8) Proton C board is not currently supported.

Proton C is a Pro Micro pin-compatible board that uses an ARM microcontroller. This is not supported because it's not using an AVR core, and it's unlikely to be supported in the future.

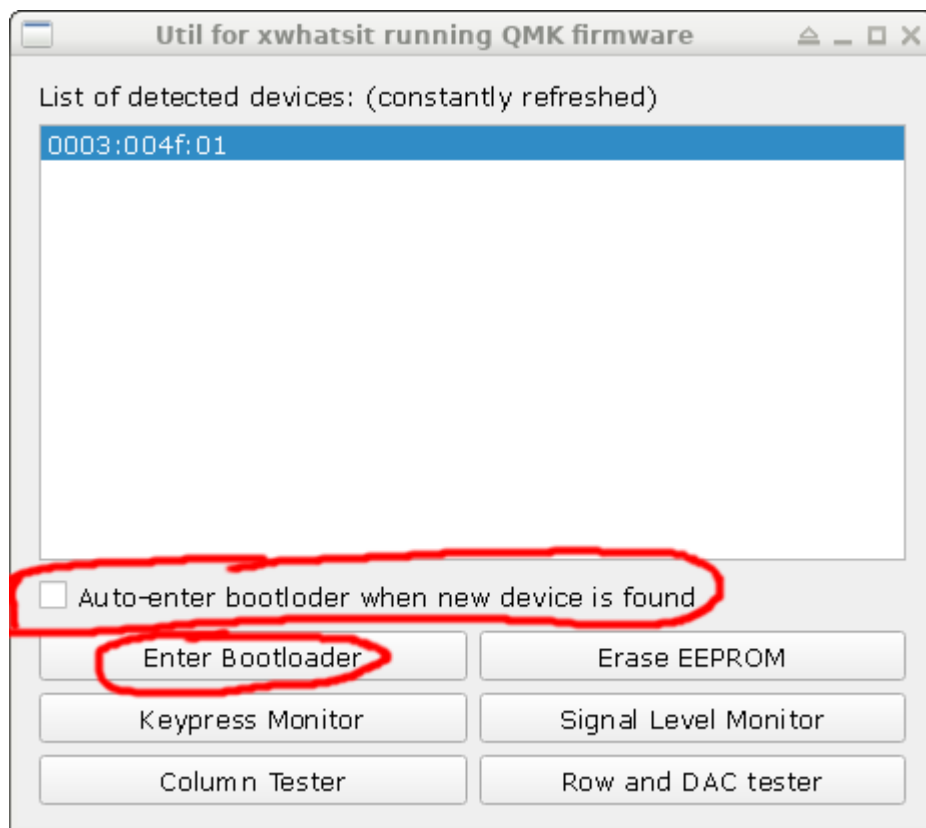
Entering bootloader mode using the host-side utility

This will only work if the flashed firmware is functional enough that it can enumerate on USB, and if it is one of the firmwares that supports boot-loader entry via USB command.

The following firmwares support bootloader entry from a USB command:

- Original Xwhatsit firmware
- Current versions of QMK firmware running on xwhatsit
 - Please keep in mind that we might disable this feature by default for future versions, because it is considered a security risk. Please check back here for updates.

The main window of the host-side utility looks like this:



If your keyboard is detected, it will show up in the list of detected devices. You can enter bootloader mode by clicking the “Enter Bootloader” button. If your keyboard is generating random keystrokes, and making it impossible to click this button, then you also have the option of checking the “Auto-enter bootloader when new device is found” checkbox, minimizing the util window (to avoid it being closed by the keyboard generating Alt-F4 or similar), and then plugging in the keyboard.

Please keep in mind that even if you use this method, Catalina-based bootloaders have a 8-second timeout, so your keyboard will show up again after 8 seconds if you don’t flash it immediately.

Entering bootloader mode using the QMK “RESET” keycode

When configuring your keymap, you have the option of placing the QMK RESET keycode on a function layer. When this keycode is pressed, the keyboard immediately enters bootloader mode.

When editing your keymap using configurator, the RESET keycode can be found on the Quantum tab:



This only works if the keyboard has the right firmware flashed, and is otherwise completely functional.

Entering bootloader mode using MAGIC_KEY_BOOTLOADER

QMK supports a feature called “Command”, which is described in detail over here:

https://beta.docs.qmk.fm/using-qmk/advanced-keycodes/feature_command

Command is currently enabled by default for all supported keyboards.

By default Command is configured for the following:

Configuration	Default
IS_COMMAND	Left-Shift + Right-Shift
MAGIC_KEY_BOOTLOADER	B
MAGIC_KEY_BOOTLOADER_ALT	ESC

So by default either the LeftShift-RightShift-B or LeftShift-RightShift-ESC commands will make the keyboard enter bootloader mode.

You can change the key combinations by redefining these in config.h, or you can disable this feature by setting “COMMAND_ENABLE = no” in rules.mk

This only works if the keyboard has the right firmware flashed, and is otherwise completely functional. If you have a firmware for a different keyboard flashed, it won’t work, because the keys are likely in different matrix positions.

Entering bootloader using “QMK Bootmagic”

This is not supported.

For explanation of what QMK Bootmagic is, see:

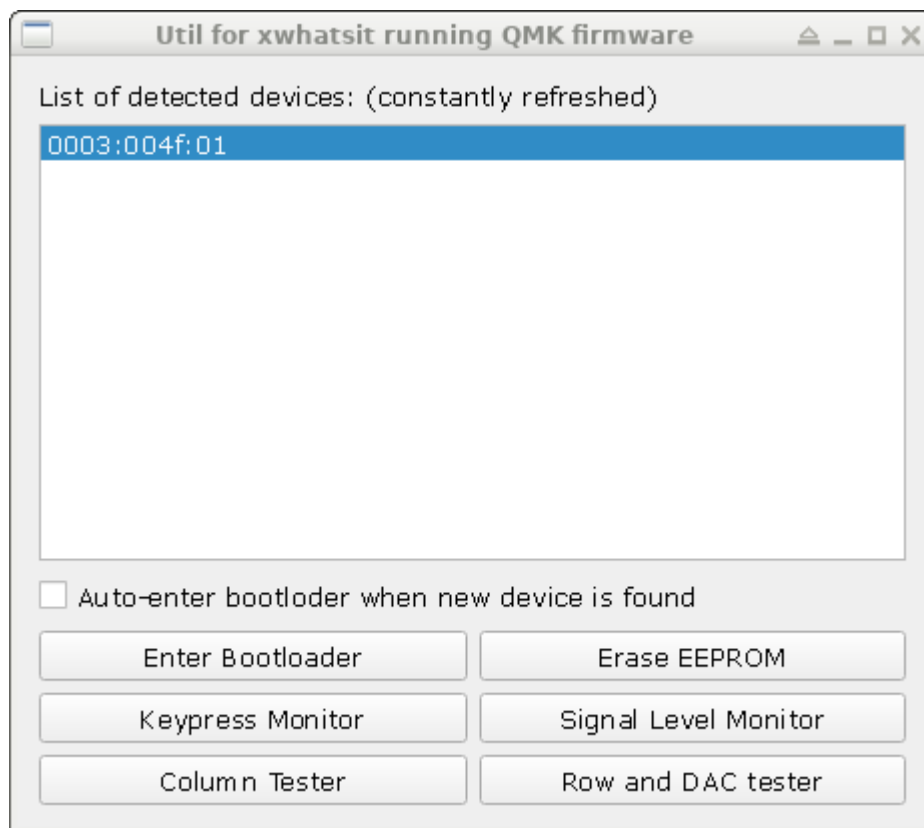
https://beta.docs.qmk.fm/using-qmk/hardware-features/feature_bootmagic

QMK Bootmagic is not supported because when the keyboard is plugged in, calibration occurs. During this time the user should not press any keys, and if the user presses the bootmagic key, it won’t be detected, precisely because calibration fails for that key.

For this reason auto-calibration and bootmagic may not be enabled at the same time in this firmware.

Getting/Building the host-side utility

This is how the main window of the host-side utility looks like:



- Linux/MacOS - you have to build it:
 - git clone http://purdea.ro/qmk_firmware/
 - (WARNING: URL IS TEMPORARY!!!)
 - cd qmk_firmware/keyboards/xwhatsit/util/util
 - Read README.md for instructions on how to build
- Windows:
 - Download from here:
 - <http://purdea.ro/volatile/util.exe>
 - (WARNING: URL IS TEMPORARY!!!)
 - If you want to build from scratch (optional)
 - We are currently cross-compiling from Linux to Windows
 - git clone http://purdea.ro/qmk_firmware/
 - (WARNING: URL IS TEMPORARY!!!)
 - cd
qmk_firmware/keyboards/xwhatsit/util/windows_crosscompile
 - Read README.md for instructions on how to build
 - We have not tried building on windows for windows. It may work out of the box, or it may require some tweaks. We're happy to accept patches.
- When running util on Linux, you may need to run it with sudo

Building the firmware

Keymap rules

Now matter how you build the firmware, there are a few keymap rules that need to be followed, due to how the auto-calibration works.

Unassigned keys that are N/A in configurator on layer 0 (or KC_NO in a .c kemap), will not be calibrated, and will be ignored. Only real keys must be calibrated. Hidden keys, that are under bigger keys, and only used for certain layouts should not be calibrated, so they should be set to N/A (KC_NO).

This results in two rules:

- All keys that are used must be assigned to something non-N/A (non-KC_NO)
- All hidden, unused keys must be assigned to N/A (KC_NO)

Supported keyboards will have various LAYOUTs available, in general there will be:

- LAYOUT_all showing all the possible sensing pads on the sensing PCB
 - This is really there for people modding their keyboards, but it's fine to use it even if you aren't modding it. You just need to keep more attention to strictly follow the above keymap rules. And you need to know which sensing pads are used underneath each key. If you use a different layout, then you don't need to know, since you will see the large key as a single key.
- LAYOUT - if there is a layout called just LAYOUT, it implies that this is the key configuration the keyboard comes with from the factory, and it's probably a single configuration it comes with.
- LAYOUT_description - you will see layouts for keyboards that can be configured by the manufacturer in various ways.

It is recommended to find the LAYOUT macro that exactly matches the keys that you have on your keyboard, and only if there is no such layout to use LAYOUT_all.

Where to build

You have two options:


- Build using the online tool, QMK configurator
 - This will be much easier, and user friendly because you don't have to worry about setting up a local build environment. However the downside is that not all advanced QMK features are available from the configurator.
 - For new users it is recommended to start out with building a firmware in the configurator. It is easy to convert a configurator .json keymap to a .c keymap for local building, if it becomes necessary.
- Build locally, from the command line
 - You will have to set up a build environment, and some programming knowledge is recommended.

Building the firmware using configurator

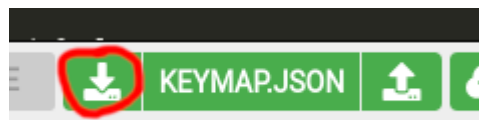
- Go to this custom configurator instance: <http://35.164.28.200:5000/>

- (WARNING: URL IS TEMPORARY!!!)
- Select your keyboard (make sure you select the right variant if you see multiple. Keyboard variants are not compatible with each other)
- Select your LAYOUT
- Configure your keymap.
 - You can drag and drop keys from the bottom using your mouse
 - You can type keys directly by clicking first on a key
 - You can click the little x icon in a key to make it N/A
 - You can create your standard momentary function layers using the





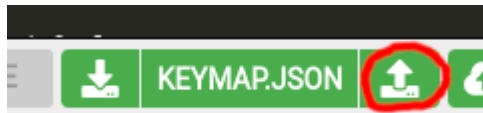
MO () square from the Quantum tab. And write the target layer into the little square.

- On deeper layers use the inverse triangle (aka KC_TRNS) over momentary function layer keys
- Save your keymap json, so you don't have to recreate it when you want to



make mods:

- Click COMPILE () on the top right corner
- If it stays on Queuing... forever, then it has frozen, and contact DT user pandrew to restart the service (happens once in a while)
- Click download FIRMWARE () on the bottom right corner
 - If you get any warnings that changes made may not be saved, ignore it. (click Leave/Continue)
- If you want to edit an existing keymap use the keymap upload button:



Building the firmware locally on the command line

- Follow QMKs instructions to set up all dependencies: https://docs.qmk.fm/#/newbs_getting_started
- After your QMK build environment is complete, switch to my temporary repo:
 - cd qmk_firmware (go into the qmk_firmware folder, wherever it ends up checked out)
 - git remote add pandrew http://purdea.ro/qmk_firmware/
 - (WARNING: ABOVE URL IS TEMPORARY!!!)
 - git fetch pandrew
 - git checkout -b wip pandrew/wip
- Example build command:
 - cd qmk_firmware
 - make xwhatsit/brand_new_model_f/f62:default
- Example build & flash command
 - cd qmk_firmware

- make xwhatsit/brand_new_model_f/f62/wcass:default:flash
- Example how to use “qmk” command line tool:
 - Run these once:
 - qmk config user.keyboard=xwhatsit/brand_new_model_f/f62/wcass
 - qmk config user.keymap=default
 - Run these every time it’s needed:
 - qmk compile
 - qmk flash
 - Example how to build a .json file downloaded from configurator
 - qmk compile path/to/file.json
 - This might leave a keymap.c file in
xwhatsit/\$(VENDOR)/\$(KEYBOARD)/keymaps/default_\$(random_characters)
 - qmk flash path/to/file.json
 - Result of qmk compile is a “hex” file in the qmk_firmware folder
- To convert to a .c keymap:
 - qmk json2c path/to/file.json

Before flashing, if switching from original xwhatsit firmware

If you are switching from original xwhatsit firmware, you probably want to ensure that you can easily revert back and not lose your settings. In order to do that you should:

- Save the “.l” layout file from the ibm_capsense_util
- Take a note of the threshold settings. These are NOT saved as part of the “.l” file.

Flashing

Before flashing please put your device into bootloader mode. (except for when flashing with the QMK build environment).

Beware that on the Pro Micro-based controllers the bootloader mode has a timeout.

There are various options for flashing firmware:

- QMK Toolbox
 - <https://qmk.fm/toolbox/>
 - Works on Windows, and MacOS
 - Can flash all types of xwhatsit controllers
 - Open Source
 - Make sure you select:
 - atmega32u2 for xwhatsit/wcass controllers
 - atmega32u4 for Pro Micro based controllers
- ATMEL FLIP
 - Works on Windows and Linux
 - Can only flash atmel-dfu bootloader devices (TODO CHECK:IS THIS TRUE?)
 - Proprietary software

- Using the QMK build environment on the command line (recommended when building firmware locally)
 - “qmk flash” command (if you have the qmk tool set up)
 - “make xwhatsit/\$(VENDOR)/\$(KEYBOARD)/\$(CONTROLLER):\$(KEYMAP):flash” command
 - If you use this with a ‘universal’ controller, I recommend running this first, until it starts waiting for a device, and then putting the device in bootloader mode.
 - Works on:
 - Linux (please make sure to follow QMKs own setup instructions. It will install udev rules for you)
 - MacOS
 - Windows (TODO CHECK:IS THIS TRUE? SHOULD BE. Does it need specific drivers?)
 - Can flash all types of xwhatsit controllers
- Using individual command-line tools:
 - “dfu-programmer” to flash atmel-dfu bootloader devices
 - Example command line on linux, to flash original xwhatsit or wcass boards:
 - sudo dfu-programmer atmega32u2 erase
 - sudo dfu-programmer atmega32u2 flash FILENAME.hex
 - “avrdude” to flash caterina bootloader devices
 - Example command line on linux:
 - sudo avrdude -p atmega32u4 -P /dev/ttyACM0 -c avr109 -U flash:w:FILENAME.hex

Common usage tips

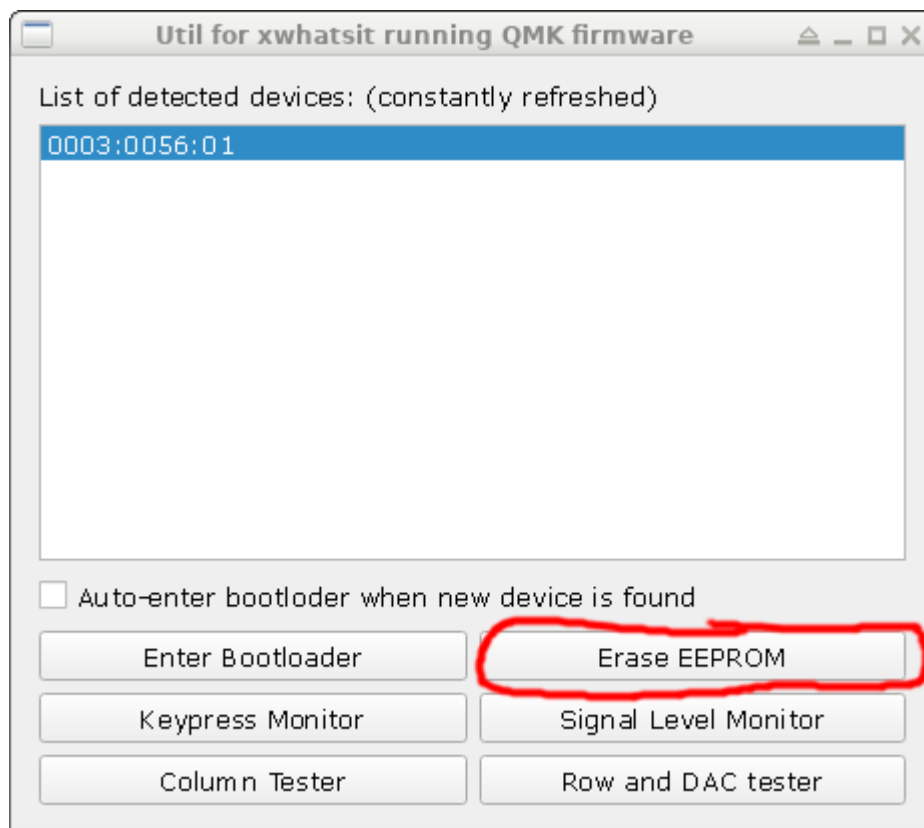
- When using multiple layers, on your target layer, in the same position as in the calling function key, make sure to set the keycode to transparent: KC_TRANSPARENT / KC_TRNS / _____ or in configurator it is the upside down triangle.
- If you have a locking capslock key, like on some displaywriter keyboards, use the KC_LCAP keycode. (If you are using configurator, place the “Any” key from the Quantum tab on your capslock key, and write “KC_LCAP” in the box)

Debugging

Erasing EEPROM

QMK stores in EEPROM the index of the base layer.

Sometimes if there is trash in EEPROM, it will think that the base layer is some non-zero layer, and it will either not generate keycodes, or it will type garbage. If this happens, it's easiest to fix this by clicking the Erase EEPROM button in the host side utility:



And then unplugging, and re-plugging the keyboard.

How Capacitive Sensing Works

TODO TODO TODO

Model (capacitances don't really follow real life)

<http://tinyurl.com/y3danzdx>

How Auto-Calibration works

TODO TODO TODO

Keypress Monitor

TODO TODO TODO

Signal Level Monitor

TODO TODO TODO

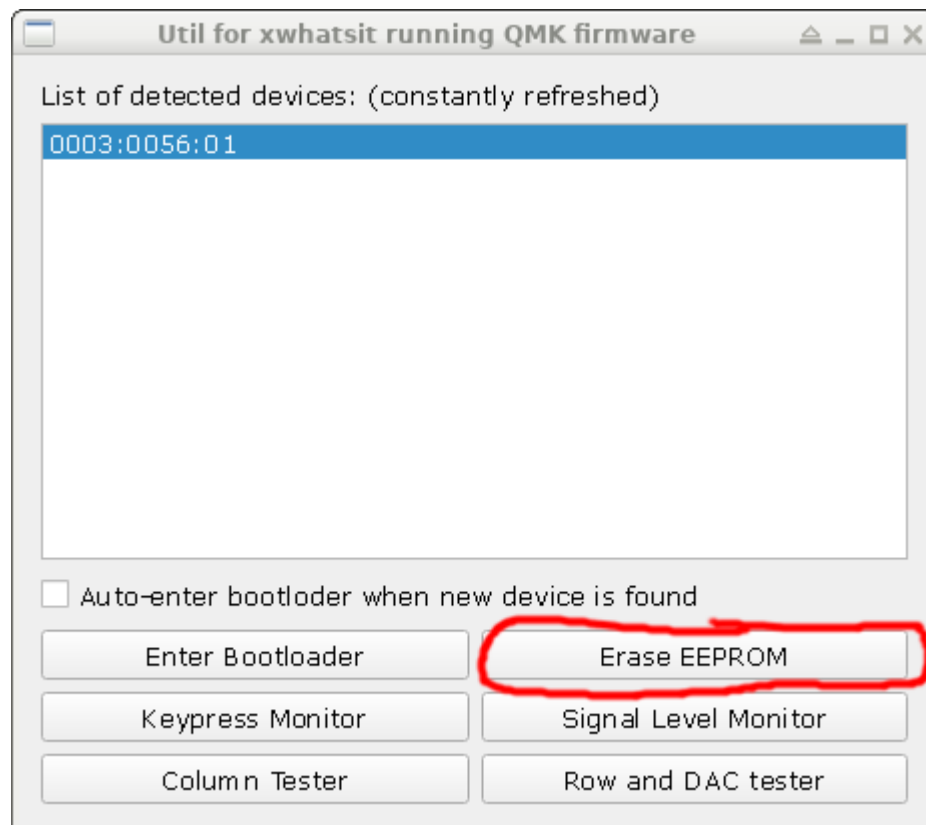
Debugging freshly-built controllers when it just doesn't work

TODO TODO TODO

Reverting back to original xwhatsit firmware

Reverting back to original xwhatsit firmware, is as simple as going into bootloader mode and flashing the original xwhatsit firmware .hex files, however there are a few things to keep in mind:

- It is recommended to erase EEPROM when going back to xwhatsit firmware. Trash left in the EEPROM could make the original xwhatsit firmware misbehave, and generate random characters, making it hard to configure. (if it's not generating lots of random keypresses, then there is no issue if you don't erase the eeprom)
 - You can erase EEPROM just before you enter the bootloader to flash back the original xwhatsit firmware, by using the host side util



And then you can click Enter Bootloader.

But keep in mind that every time QMK firmware boots, it might re-initialise it's data structures in EEPROM, if it finds them in an invalid state, so if you don't succeed flashing immediately after Erasing EEPROM, and entering bootloader, you will have to erase the EEPROM again. This can be a little annoying when you are flashing a device that has a bootloader with a timeout, if you are not fast enough to flash it before the timeout expires.

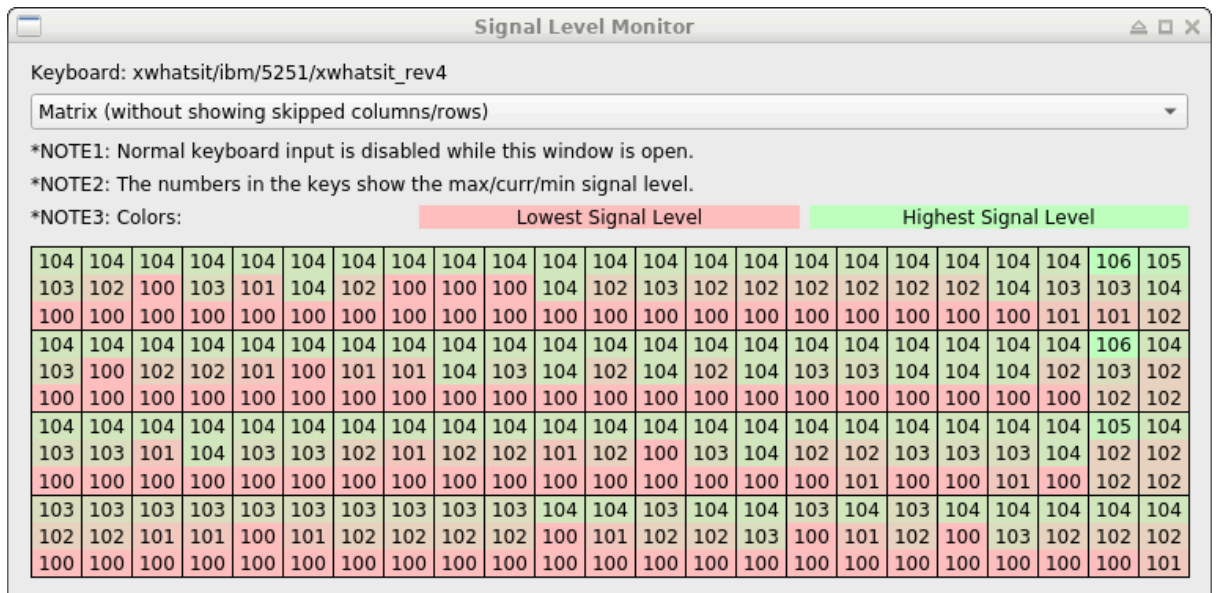
- Once you have flashed the original xwhatsit firmware, if you are not using calibration, you should set the proper threshold in `ibm_capsense_util`.
 - Keep in mind that by default the `ibm_capsense_util` doesn't allow keyboard input, so you can't type in a new threshold value using a different keyboard. This is for safety, if random keypresses are generated by the firmware, to not corrupt the `ibm_capsense_util`. In this state only the mouse can modify the threshold values (by clicking the arrows, or using the mouse wheel). You can

disable this state, by going to the “Tools” menu, and clicking “GUI keyboard unlock”. That way you can type in a threshold value using another keyboard.

- And then you can load your saved .l layout file

Testing newly-built controllers, without keyboards

- Test the USB port, apply some pressure in all directions, make sure that it doesn't loose connection to the host.
 - If using Linux, run “sudo dmesg -w”. If it doesn't work on your distro, look here for more ways to look at live dmesg output:
<https://unix.stackexchange.com/questions/95842/how-can-i-see-dmesg-output-as-it-changes>
 - For other OSes I'm not sure what would be the best equivalent, on windows maybe looking at Device Manager, and listening for USB connection/disconnection sounds.
 - If using one of the “universal” controllers with a Pro Micro, I recommend testing the USB port before soldering on the Pro Micro
 - For the Pro Micro you can also look at the power led, and make sure it doesn't blink while applying some stress to the USB connector. This won't test all the pins of the connector, but sometimes can catch a wonky one.
- If using one of the “universal” controllers that has an 5V Pro Micro, make sure the J1 jumper is shorted, and measure VCC to GND. Ideally with a good cable and good USB host it should show >4.9V. Take a note of the measured value, we will use it later.
- Flash one of the compatible QMK firmwares onto the board, preferably one that uses most of the columns, for example:
 - For beamspring, flash the 5251 or 3101_3727_3278_87key firmware.
 - For model F flash the fat or the f122 firmware
 - For displaywriter flash the displaywriter firmware
 - When testing the through_hole that can be used for various keyboards, flash the 4978 firmware - this one uses the most columns and rows.
- Open “Signal Level Monitor” without a keyboard attached, and without touching any of the pins. Switch it to matrix mode. The resulting signal levels should look like this for an original xwhatsit or wcass xwhatsit:

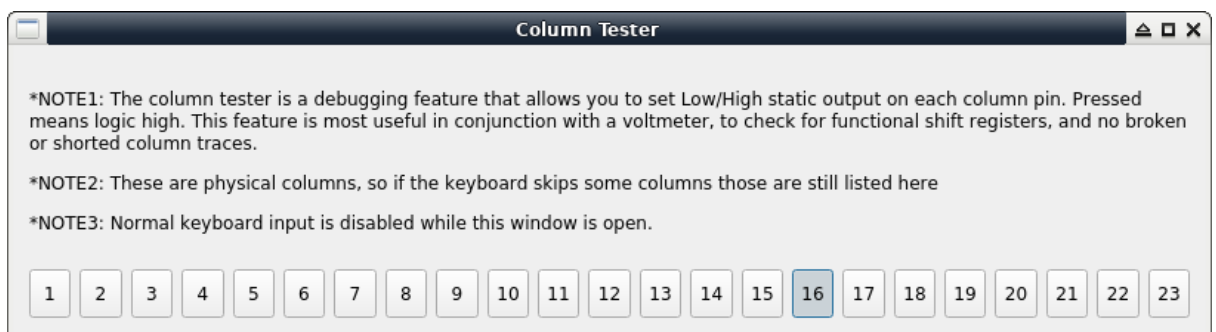


All the values should sit around “100”, you should not see major variation between rows.

In the case of a “universal” controller, it uses a DAC with more bits, so the values will sit around “300..400”. (Although I have given a big range here, I expect the values to be very similar to each other. If you see some rows look out-of-place, it’s possible that there is still flux residue left over them, and in that case I recommend cleaning the boards.)

If you see some rows having significantly higher/lower signal value then some others, and the difference is significant, that may suggest that there is flux on the board around the rows, that has not been cleaned away properly. Flux can be somewhat conductive, and cause some inconsistency here. For this reason I suggest doing this test after soldering everything onto the board, including pin headers, and especially beamspring connectors (if this is a beamspring board). I recommend cleaning of all flux no matter what you see in the signal level monitor, and testing after it has dried.

- Using the column tester, at the very least make sure that the last column is accessible:

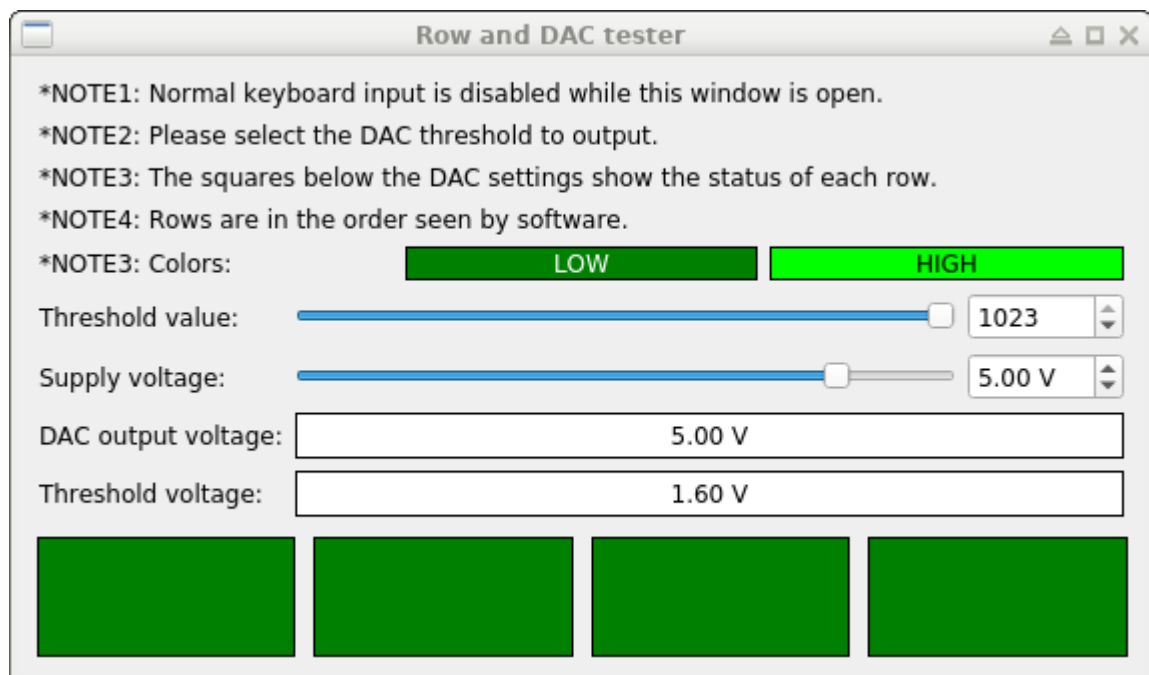


For example for a model F controller, please toggle column **16**, and measure with a multimeter the column and see it change between ~5V and ~GND.

This verifies that the shift register chain is working, and that the controller can talk to the shift registers. For beamspring controllers please test column **23**.

Optionally you may test *all* the columns, to check for soldering issues.

- To check the sensing circuitry, open up Row and DAC tester:



1. Set Supply voltage to what you have measured as one of the previous steps.
2. Lower the threshold value until the DAC output voltage is not minimum and not maximum. (ideally just under 2V if you don't have an auto-ranging multimeter)
3. While the Row and DAC tester is open, measure the DAC output voltage to GND and threshold voltage to GND in the circuit. They should closely match the calculated values.
 - Some of the universal SMD controllers have test pads to measure these.
 - If your controller has no test pads, measure these on the DAC output pin, and on the comparator minus pin.
4. Set the threshold value to 0, all the green squares should light up. This means the comparator sees a high value.
 - While the threshold stays at 0, you can test each row, by shorting the row to gnd. The shorted rows should go dark green.

Using expansion headers

Solenoids & solenoid drivers

Support for solenoids is enabled by default.

For more information, please see this variant of the haptic feedback documentation:

https://github.com/qmk/qmk_firmware/blob/develop/docs/feature_haptic_feedback.md

Not all of the information from this page is relevant to us, if you are using a normal solenoid, all information regarding DRV2605L can be ignored, unless you have a custom modified keyboard that actually uses a DRV2605L.

It is recommended to add the following keycodes to your keymap:

Keycode	Description
HPT_TOG	Turns on/off the solenoid. (You can also add HPT_ON/HPT_OFF as separate keys)
HPT_RST	This reset every solenoid-related setting to the default, which is: Solenoid on, Solenoid buzz off, Dwell time = default dwell time (currently 4ms)
HPT_BUZ	This toggles the solenoid buzzing functionality. When buzzing is enabled, for each keypress the solenoid will generate a train of pulses. You might not notice this with a low dwell time. This is designed for a high dwell time, please see the above linked haptic feedback documentation. There is no equivalent to solenoid buzz in original xwhatsit firmware, most people will usually keep this feature off.
HPT_DWLI	Clicking this keycode increases dwell time by SOLENOID_DWELL_STEP_SIZE
HPT_DWLD	Clicking this keycode decreases dwell time by SOLENOID_DWELL_STEP_SIZE

These keycodes can be added in the configurator, by placing the “Any” key from the “Quantum” tab, and then typing “HPT_...” into the little box inside that Any key.

All of the solenoid settings are saved in eeprom. Erasing the EEPROM from util.exe, or corrupting it in some other way means that everything will go back to the default values, as if HPT_RST was pressed. Upgrading the firmware will normally keep previous settings.

Dwell time is equivalent to the concept of “Extend Time” from original xwhatsit firmware. The solenoid is energized for this amount of time for the click.

The “Retract time” from original xwhatsit firmware is a hold-off period after the solenoid is extended, that allows the spring built into the solenoid to return the core to its resting position, where it’s ready to click again. In QMK at preset, retract time is always zero. This means that if you type very fast, a second click that comes in right after the coil has stopped being energized for the first click, the second click might be weaker, or not even heard.

Another difference between original xwhatsit firmware and qmk for xwhatsit controllers, is that the original xwhatsit firmware has a click “queue”. If your extend-time, and retract-time are large and you type very fast, and a click can’t be executed immediately it is added to the queue to be executed later. This means that clicks are never missed, but clicks are not well correlated in time to the keypress. This queuing functionality is not currently supported in QMK firmware.

When selecting dwell time, it is recommended to select a dwell time that is as small as possible, but still clicks the solenoid so that it sounds nice. It has been found that the current minimum of 4ms works well with original solenoids from IBM. The DWLI, and DWLD keycodes are round-robin, so if the current setting is equal to the minimum, and DWLD is clicked, the solenoid dwell time wraps around to the maximum setting.

The original IBM solenoids are configured for a very short throw distance. When selecting a non-original solenoid to buy, please be aware that throw distance will affect the loudness, and the sluggishness of the solenoid, there is a tradeoff:

- High throw distance will make more noise, but requires a higher dwell time setting, and so it might not behave well, when typing really fast.
- Low throw distance will make less noise, but it will work with a low dwell time setting, so it is much faster, and allows you to type very fast.

It is recommended to buy solenoids which have adjustable throw distance in some way (for example an L-bracket that has an adjustable position, that limits the motion of the internal core), or otherwise if it doesn't come with it from the factory, it is recommended to build a bracket for the solenoid.

All the boards currently supported by QMK firmware for xwhatsit, requires an original xwhatsit solenoid driver, or a compatible, one, like the newly designed JLCPCB-assemblable solenoid driver from here:

<https://deskthority.net/viewtopic.php?f=7&t=24598>

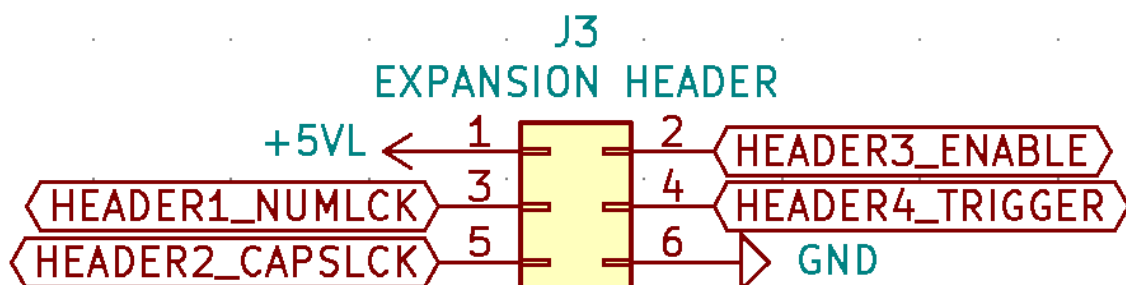
Lock lights

Lock light pins available on all controllers

Right now QMK firmware does not support the same combinations for expansion header use, as the original xwhatsit firmware supports.

Right now solenoid support is enabled by default, consuming two pins of the expansion header (enable and trigger). Two pins are used for power, and the remaining two pins can be used for lock lights, and by default these are configured to be Num Lock and Caps Lock. The normal expansion header has no built-in current limiting resistors, so you will have to add those.

Please see the following pinout:



To make use of pins 3 and 5, you should add a current limiting resistor, and a led connected in series, from pin 3/5 to pin 6. The anode side of the led (the longer lead, or the pin that is connected to the smaller post inside the led) should face towards pin 3/5.

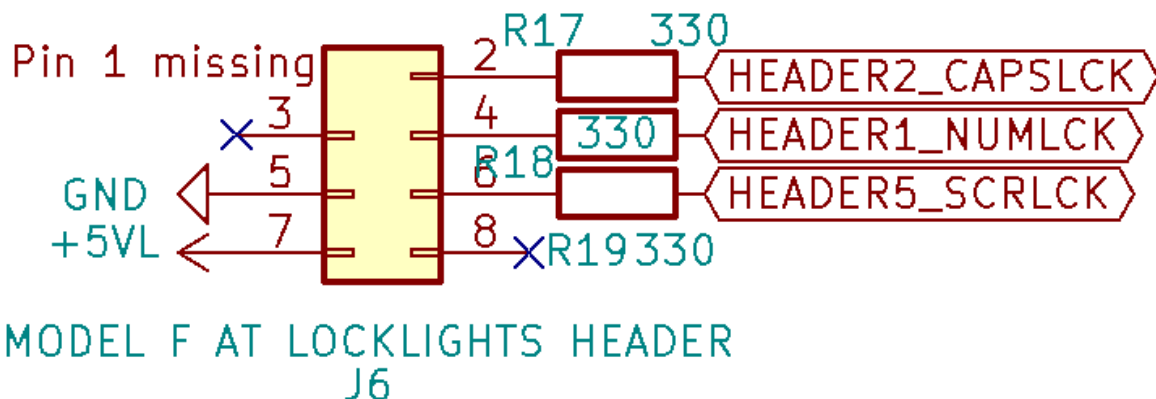
If solenoid is not in use

If solenoid is not in use, it can be disabled, and the control pins can be reused, only if compiling your firmware locally, by editing config.h:

- comment out `#define SOLENOID_ENABLE_PIN`
- comment out `#define SOLENOID_PIN`, and all other `#define SOLENOID_*` lines.
- edit/add/remove `#define LED_*_LOCK_PIN` lines
- Polarity can be changed to active low, by adding `#define LED_*_LOCK_ACTIVE_LOW`

Additional lock light pins available on the “standard” SMD Model F controllers:

The following additional lock light header is available on the standard SMD Model F controller:



This header is compatible with the IBM Model F AT keyboard lock light board and lock light cable. You should solder in a dual-row 90-degree/right-angle pin header, with the plastic pieces on the long side. You will also need to extract one of the pins out of the plastic.

When using this header, but not with the Model F AT locklights, please keep in mind that current limiting resistors have been added to this pin header. These pins continue being active high by default, and that the orientation of the leds should be anode facing pin 2/4/6, and cathode facing pin 5.

Additional lock light pins available on the “kishsaver” SMD Model F controllers:

Similarly to the pins available on the “standard” SMD Model F controller, the same signals are available on the “kishsaver” class controller, but these are now a single row of holes, and wires can be soldered to these. It is not recommended to mount pin headers in these holes, because of the lack of space in the keyboard:

J7
Conn_01x05

