QMK Keymap Overview

Keymap Overview

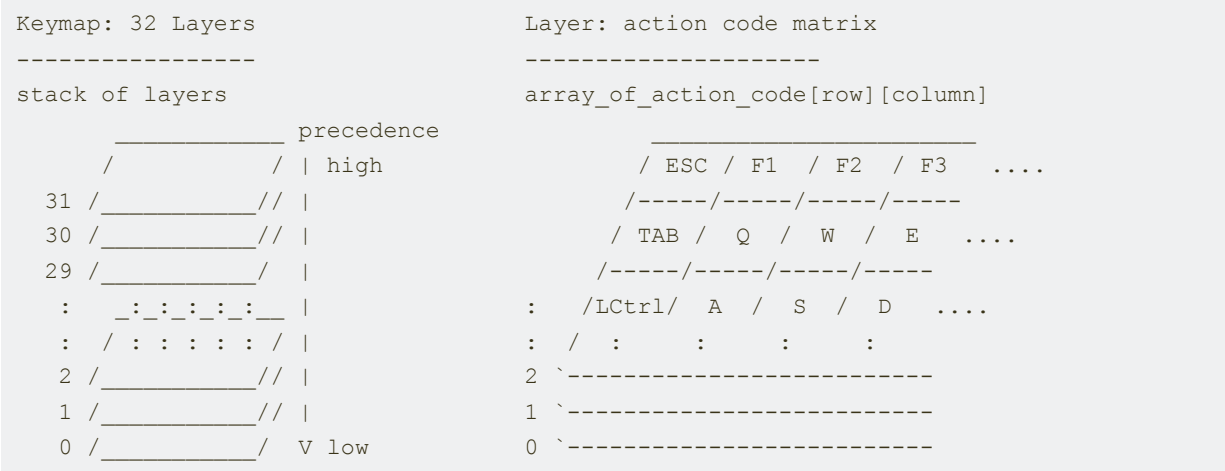
QMK keymaps are defined inside a C source file. The data structure is an array of arrays. The outer array is a list of layer arrays while the inner layer array is a list of keys. Most keyboards define a `LAYOUT()` macro to help you create this array of arrays.

Keymap and Layers

In QMK, **`const uint16_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS]`** holds multiple **layers** of keymap information in **16 bit** data holding the **action code**. You can define **32 layers** at most.

For trivial key definitions, the higher 8 bits of the **action code** are all 0 and the lower 8 bits holds the USB HID usage code generated by the key as **keycode**.

Respective layers can be validated simultaneously. Layers are indexed with 0 to 31 and higher layer has precedence.



Sometimes, the action code stored in keymap may be referred as keycode in some documents due to the TMK history.

Keymap Layer Status

The state of the Keymap layer is determined by two 32 bit parameters:

- **`default_layer_state`** indicates a base keymap layer (0-31) which is always valid and to be referred (the default layer).
- **`layer_state`** has current on/off status of each layer in its bits.

Keymap layer '0' is usually the `default_layer`, with other layers initially off after booting up the firmware, although this can configured differently in `config.h`. It is useful to change `default_layer` when you completely switch a key layout, for example, if you want to switch to Colemak instead of Qwerty.

```
Initial state of Keymap          Change base layout
-----------------------          ------------------

  31                               31
  30                               30
  29                               29
   :                                :
   :                                :   _____________
   2   _____________                2  /             /
   1  /             /              ,->1 /_____________/
,->0 /_____________/               |  0
|                                  |
`--- default_layer = 0             `--- default_layer = 1
     layer_state   = 0x00000001         layer_state   = 0x00000002
```

On the other hand, you can change `layer_state` to overlay the base layer with other layers for features such as navigation keys, function keys (F1-F12), media keys, and/or special actions.

```
Overlay feature layer
---------------------      bit|status
       _____________       ---+------
  31  /             /      31 |   0
  30 /_____________// -----> 30 |   1
  29 /_____________/  -----> 29 |   1
   :                          : |   :
   :   _____________          : |   :
   2  /             /         2 |   0
,->1 /_____________/  ----->   1 |   1
|  0                           0 |   0
|                                +
`--- default_layer = 1           |
     layer_state   = 0x60000002 <-'
```

Layer Precedence and Transparency

Note that ***higher layers have higher priority within the stack of layers***. The firmware works its way down from the highest active layers to look up keycodes. Once the firmware locates a keycode other than `KC_TRNS` (transparent) on an active layer, it stops searching, and lower layers aren't referenced.

```
       _____________
      /             /  <--- Higher layer
     /  KC_TRNS    //
    /_____________//   <--- Lower layer (KC_A)
    /_____________/

In the above scenario, the non-transparent keys on the higher layer would be usable,
but whenever `KC_TRNS` (or equivalent) is defined, the keycode (`KC_A`) on the lower
level would be used.
```

Note: Valid ways to denote transparency on a given layer:

- `KC_TRANSPARENT`
- `KC_TRNS` (alias)
- `_______` (alias)

These keycodes allow the processing to fall through to lower layers in search of a non-transparent keycode to process.

Anatomy of a `keymap.c`

For this example we will walk through an older version of the default Clueboard 66% keymap. You'll find it helpful to open that file in another browser window so you can look at everything in context.

There are 2 main sections of a `keymap.c` file you'll want to concern yourself with:

- The Definitions
- The Layer/Keymap Datastructure

Definitions

At the top of the file you'll find this:

```
#include QMK_KEYBOARD_H

// Helpful defines
#define GRAVE_MODS
(MOD_BIT(KC_LSFT)|MOD_BIT(KC_RSFT)|MOD_BIT(KC_LGUI)|MOD_BIT(KC_RGUI)|MOD_BIT(KC_LALT)|MOD_BIT(K


/* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
 *  You can use _______ in place for KC_TRNS (transparent)   *
 *  Or you can use XXXXXXX for KC_NO (NOOP)                  *
 * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * */

// Each layer gets a name for readability.
// The underscores don't mean anything - you can
// have a layer called STUFF or any other name.
// Layer names don't all need to be of the same
// length, and you can also skip them entirely
// and just use numbers.
enum layer_names {
    _BL,
    _FL,
    _CL,
};
```

These are some handy definitions we can use when building our keymap and our custom function. The `GRAVE_MODS` definition will be used later in our custom function, and the following `_BL`, `_FL`, and `_CL` defines make it easier to refer to each of our layers.

Note: You may also find some older keymap files may also have a define(s) for `_______` and/or `XXXXXXX`. These can be used in place for `KC_TRNS` and `KC_NO` respectively, making it easier to see what keys a layer is overriding. These definitions are now unnecessary, as they are included by default.

Layers and Keymaps

The main part of this file is the `keymaps[]` definition. This is where you list your layers and the contents of those layers. This part of the file begins with this definition:

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

After this you'll find the layer definitions. Typically you'll have one or more "base layers" (such as QWERTY, Dvorak, or Colemak) and then you'll layer on top of that one or more "function" layers. Due to the way layers are processed you can't overlay a "lower" layer on top of a "higher" layer.

`keymaps[][MATRIX_ROWS][MATRIX_COLS]` in QMK holds the 16 bit action code (sometimes referred as the quantum keycode) in it. For the keycode representing typical keys, its high byte is 0 and its low byte is the USB HID usage ID for keyboard.

> TMK from which QMK was forked uses `const uint8_t PROGMEM keymaps[][MATRIX_ROWS][MATRIX_COLS]` instead and holds the 8 bit keycode.

Base Layer

Here is an example of the Clueboard's base layer:

```
[_BL] = LAYOUT(
    F(0),    KC_1,    KC_2,   KC_3,   KC_4,   KC_5,   KC_6,   KC_7,   KC_8,   KC_9,
KC_0,     KC_MINS,  KC_EQL,   KC_GRV,  KC_BSPC,          KC_PGUP,
    KC_TAB,  KC_Q,    KC_W,   KC_E,   KC_R,   KC_T,   KC_Y,   KC_U,   KC_I,   KC_O,
KC_P,     KC_LBRC,  KC_RBRC,  KC_BSLS,                   KC_PGDN,
    KC_CAPS, KC_A,    KC_S,   KC_D,   KC_F,   KC_G,   KC_H,   KC_J,   KC_K,   KC_L,
KC_SCLN,  KC_QUOT,  KC_NUHS,  KC_ENT,
    KC_LSFT, KC_NUBS, KC_Z,   KC_X,   KC_C,   KC_V,   KC_B,   KC_N,   KC_M,
KC_COMM, KC_DOT,   KC_SLSH,  KC_INT1,  KC_RSFT,          KC_UP,
    KC_LCTL, KC_LGUI, KC_LALT, KC_INT5,          KC_SPC,KC_SPC,
KC_INT4,  KC_RALT,  KC_RCTL,  MO(_FL), KC_LEFT, KC_DOWN, KC_RGHT
),
```

Some interesting things to note about this:

- The layer is defined using the LAYOUT macro, traditionally defined in the keyboard's `.h` file.
- The LAYOUT macro takes a single list of keycodes, but we have written it in the C source using embedded whitespace and newlines to visualize where each key is on the physical device.
- The LAYOUT macro hides and handles the mapping to the hardware's key scan matrix.
- Plain keyboard scancodes are prefixed with KC_, while "special" keys are not.
- The upper left key activates custom function 0 (`F(0)`)
- The "Fn" key is defined with `MO(_FL)`, which moves to the `_FL` layer while that key is being held down.

Function Overlay Layer

Our function layer is, from a code point of view, no different from the base layer. Conceptually, however, you will build that layer as an overlay, not a replacement. For many people this distinction does not matter, but as you build more complicated layering setups it matters more and more.

```
[_FL] = LAYOUT(
    KC_GRV,  KC_F1,   KC_F2,  KC_F3,  KC_F4,  KC_F5,  KC_F6,  KC_F7,  KC_F8,  KC_F9,
KC_F10,   KC_F11,   KC_F12,   _______, KC_DEL,          BL_STEP,
    _______, _______,
```

```
_______,_______,_______,_______,_______,_______,KC_PSCR,KC_SCRL, KC_PAUS,  _______,
_______,  _______,                  _______,
    _______, _______,
MO(_CL),_______,_______,_______,_______,_______,_______,_______, _______,  _______,
_______,  _______,
    _______, _______,
_______,_______,_______,_______,_______,_______,_______,_______, _______,  _______,
_______,  _______,          KC_PGUP,
    _______, _______, _______, _______,        _______,_______,
_______,  _______,  _______,  MO(_FL), KC_HOME, KC_PGDN, KC_END
),
```

Some interesting things to note:

- We have used our _____ definition to turn `KC_TRNS` into _____. This makes it easier to spot the keys that have changed on this layer.
- While in this layer if you press one of the _____ keys it will activate the key in the next lowest active layer.

Nitty Gritty Details

This should have given you a basic overview for creating your own keymap. For more details see the following resources:

- Keycodes
- Keymap FAQ

We are actively working to improve these docs. If you have suggestions for how they could be made better please file an issue!

✏️ Edit this page