# Link Select Vial GUI features

1. Porting guide
2. Select Vial GUI features (for size)

By default Vial enables every supported feature. As a result, on low-powered chips such as AVR series you might experience running out of flash memory (compiled size), RAM, or EEPROM.

```
quantum/dynamic_keymap.c:102:1: error: static assertion failed: "Dynamic keymaps are configured to use more EEPROM than is available."
_Static_assert(DYNAMIC_KEYMAP_EEPROM_MAX_ADDR >= DYNAMIC_KEYMAP_MACRO_EEPROM_ADDR + 100, "Dynamic keymaps are configured to use more EEPROM than is available.");
```

```
 * The firmware is too large! 30768/28672 (2096 bytes over)
 [ERRORS]

make[1]: *** [tmk_core/rules.mk:469: check-size] Error 1
make: *** [Makefile:530: yd60mq:vial] Error 1
Make finished with errors
```

If that happens to your keyboard, you can try utilizing of the options below to make the firmware fit. Depending on the type of keyboard (full-size/split/macropad etc), some options simply aren’t useful, and others may be entirely neccesary for it to function as intended.

This also means that some older designs with less powerful MCU’s like the classic AVR line in the popular Pro Micro controllers, may need to disable features or reduce the amount of slots configured in some features to make the firmware fit into compiled memory and not run out of EEPROM or RAM.

Besides compiled size, all features share the available EEPROM, and if you desire more than the standard amount of slots for one, you may need to disable one feature entirely or reduce the alloted memory to it for balance.

## QMK Settings

QMK Settings allows you to configure settings for all the other GUI features, like Tapping term/Permissive hold/Tapping delay for Tap Dance, and Mouse key timing, Auto shift behaviour etc. Without it, you are missing out on a lot of Vial’s actual functionality.

### Configure memory usage

To turn off this feature and reduce compiled firmware size, RAM and EEPROM usage, add the following line to your `keymaps/vial/rules.mk`:

```
QMK_SETTINGS = no
```

## Tap Dance

Tap Dance is the GUI equivalent to QMK's Tap Dance and can do pretty much what it does, with minor differences as you assign it in GUI rather than code. Tap the comma key, and you get a comma. Tap it twice and you get a semicolon.

### Configure memory usage

By default, the number of available Tap Dances are calculated from the amount of EEPROM your controller have. To reduce EEPROM usage or to select one feature over another, you can define the following in your `config.h`:

```
#define VIAL_TAP_DANCE_ENTRIES x
```

Where `x` is the number of slots you desire.

To turn off this feature completely, and also reduce compiled firmware size, EEPROM and RAM usage, add the following line to your `keymaps/vial/rules.mk`:

```
TAP_DANCE_ENABLE = no
```

## Combos

Combos are the GUI equivalent to QMK's Combos and is a chording solution for adding custom actions. It lets you hit multiple keys at once and produce a different effect. For instance, hitting A and B within the combo term would hit ESC instead, or have it perform even more complex tasks, all configurable in the GUI.

### Configure memory usage

By default, the number of available Combos are calculated from the amount of EEPROM your controller have. To reduce EEPROM usage or to select one feature over another, you can define the following in your `config.h`:

```
#define VIAL_COMBO_ENTRIES x
```

Where `x` is the number of slots you desire.

To turn off this feature completely, and also reduce compiled firmware size, EEPROM and RAM usage, add the following line to your `keymaps/vial/rules.mk`:

```
COMBO_ENABLE = no
```

## Key Overrides

Key overrides is the GUI equivalent to QMK's Key Overrides and allows you to override modifier-key combinations to send a different modifier-key combination or perform completely custom actions.

Don't want shift + 1 to type ! on your computer? Use a key override to make your keyboard type something different when you press shift + 1. The general behavior is like this: If modifiers w + key x are pressed, replace these keys with modifiers y + key z in the keyboard report.

### Configure memory usage

By default, the number of available Key Overrides are calculated from the amount of EEPROM your controller have. To reduce EEPROM usage or to select one feature over another, you can define the following in your `config.h`:

```
#define VIAL_KEY_OVERRIDE_ENTRIES x
```

Where `x` is the number of slots you desire.

To turn off this feature completely, and also reduce compiled firmware size, EEPROM and RAM usage, add the following line to your `keymaps/vial/rules.mk`:

```
KEY_OVERRIDE_ENABLE = no
```

## Reducing dynamic keymap layers

By default Vial is configured to have 4 keymap layers, To reduce EEPROM usage, you can define the following in your `config.h` file:

```
#define DYNAMIC_KEYMAP_LAYER_COUNT 2
```

## Enable LTO

To enable LTO, add the following line to your `keymaps/vial/rules.mk`:

```
LTO_ENABLE = yes
```

LTO makes the compiler work harder when optimizing your code, resulting in a smaller firmware size.

Information

Using LTO can in rare situations expose buggy code in a way which could break certain firmware functionality. Make sure to test your keyboard firmware throughly after enabling this option.

## Using a different bootloader

**If you have already done all of the above, and you still see this message trying to compile your firmware on an AVR such as the popular Atmega32u4 (used on Pro Micro), it might be time to break out of the mould a little.**

Almost all of these controllers are delivered with the Caterina bootloader, and while this bootloader is very stable and plain **just works!** in all situtations, it’s also fairly old code and quite large, using ~3500 bytes of your codespace. Most of it for features you will never really use.

Changing your bootloader to a more modern, more slim-lined one that does exactly what is needed and nothing else, can save you a whole lot of codespace for your keymap and functions.

### Potential drawbacks

**Changing the bootloader makes your firmware somewhat ‘non-standard’**, so sharing them means the other user also have to swap to your preffered bootloader. It also requires you to use an ISP programmer (or another arduino) to flash the controller with the bootloader once, before it can be used as normal flashing the code over USB. (This is very similar to recovering the bootloader in the event of a bricked Pro Micro).

### What bootloader is recommended?

There are several that should work and reduce the size to various amounts, some adding functions you might like, some being slimmed down.

- LUFA-DFU is a popular one that is slightly smaller than Caterina, and adds features (size depends on what features you enable).
- QMK-DFU is a fork maintained by the QMK team and also adds features.
- nanoBoot is a *tiny* 512 bytes HID bootloader, that does exactly *one* thing, allowing you to hold reset on plug-in, and enter the bootloader/flashing, in the exact same way as Caterina does.