LESSON · SCREEN DESIGNER AND ARDUINO CODE

Writing for the display boards

Revision 1.0 · 21 November 2026

A sketch for a touch screen made in SimuCraft has two authors. The screen designer writes the screen: every button, label and picture, the board's set-up, Wi-Fi, the weather, the brightness. You write what makes it yours: what happens when a button is pressed, where the numbers come from, what the board does every second. This lesson shows exactly where the line between the two runs, how they meet in one sketch, and how to write your part so the designer never gets in your way, and you never get in its.

What you need

- **SimuCraft for Windows**, 0.6.5 beta or later. Everything up to step 11 also works in the web version; putting the sketch on a real board needs the desktop app.
- A little C or Arduino: variables, `if`, functions. You don't need to know LVGL, the graphics library underneath; this lesson gives you the dozen calls you will use.
- Optional: a board to try it on. The examples here are the Guition 7" ESP32-P4, but the same rules hold for every board with a screen designer: the 4.3" Guition, the M5Stack Tab5, Core2, Dial and Core Ink, the MaTouch round displays, the T-Display, the CYD, the Freenove and the GIGA Display.
- About two hours. Read The Guition ESP32-P4 displays first if you have one of those boards; it covers the hardware and uploading.

1. Two authors, one sketch

Open **Examples** → **Guition 7" P4 display** → **Weather station**, and click **Design screen** on the board. The screen designer opens with the station's two screens. Everything you see there, the designer will write as C++: it is the author of the screen. Close it and open the board's **Code**: that is the sketch the board runs, and the designer wrote most of it.



The weather station in the screen designer. Left, the widgets you can add; middle, the screen as the board will show it; right, the properties of what is selected.

The sketch is one file, but it is written by two hands, and each line belongs to one of them:

The heading, the #includes, the board's set-up
jc_begin(): the screen, its touch chip, LVGL on them
the backlight and the screen's start-up table

Every widget's name: `lv_obj_t * ui_place;`
and the designer's own variables: `extern float ui_t;`

// USER CODE BEGIN globals
your variables, #includes and functions

Each widget's handler: `ui_next_clicked(e)`
what you chose under "When it is pressed" (Go to screen...)

```
// USER CODE BEGIN ui_next_clicked: your part of it
```

Wi-Fi, the time, the weather, ui_update(), ui_create()
builds the screens; shows the readings five times a second

setup(): Serial, jc_begin(), ui_create(), Wi-Fi...

```
// USER CODE BEGIN setup: your start-up
```

loop(): lv_timer_handler(), the weather, ui_update()

```
// USER CODE BEGIN loop: your work, a little at a time
```

written by the designer, rewritten after every change

yours: kept, word for word, every time it writes again

That one rule is the whole contract: `designer` lines are rewritten every time you change the design, so anything you type there is lost; `your` lines, between `// USER CODE BEGIN` and `// USER CODE END`, are kept exactly as you wrote them.

> A good test for where something belongs: **if you can set it in the designer, set it there.** Positions, sizes, fonts, colours, which screen a button goes to, what a label shows: all designer. Your code is for what the designer can't know: your sums, your data, your decisions.

2. The sketch, top to bottom

Here is the weather station's sketch in the order the board reads it, with who wrote each part. Open its Code window beside this lesson and scroll along.

The heading

A comment the designer writes for when the sketch leaves SimuCraft: the fonts to turn on in `lv_conf.h`, the libraries (LVGL 9, GFX Library for Arduino), and the board settings for the Arduino IDE (ESP32P4 Dev Module, PSRAM enabled, 16 MB flash). In SimuCraft you can ignore it: the upload sets all of that for you.

The #includes and the board's set-up

```
#include <Arduino_GFX_Library.h>
#include <lvgl.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <ArduinoJson.h>
#include <Preferences.h>   // the Wi-Fi networks set up on the screen, kept in flash
#include "ui_assets.h"     // the pictures and fonts
```

Only the libraries your design needs are included: `HTTPClient` and `ArduinoJson` because the design shows the weather, `Preferences` because of the Wi-Fi screens. Then come the board's pins (`TFT_BL 23` for the backlight, the touch chip on I2C pins 7 and 8), the screen's start-up table and `jc_begin()`, which starts the screen, the touch chip and LVGL. Every Guition sketch has the same block; you never need to touch it.

The widgets' names

```
lv_obj_t * ui_main;      // the Weather screen
lv_obj_t * ui_place;     // the town, top left
lv_obj_t * ui_time;
lv_obj_t * ui_next;      // the "Next 24 h" button
...
```

One line for every widget and every screen, with the name you see in the designer as **Name in the code**. They are declared here, at the top, so your code anywhere below can use them.

The designer's own variables

```
// the designer's own variables (set further down), named here so your code can use them
extern String ui_town;
extern float ui_lat, ui_lon;
extern float ui_t, ui_tmax, ui_tmin, ui_rain, ui_wind, ui_wdir, ui_uv;
extern int ui_wcode, ui_wday;
extern unsigned long ui_weather_at;
```

The weather the designer fetches, the town and its position: announced here with `extern` and filled in further down. That's how your code, which comes next, can read `ui_t` (the temperature) before the designer's weather code has been written.

Your globals

```
// USER CODE BEGIN globals
// the weather from Open-Meteo (free, no key): now, every hour for two days, and the week
const char* WX_TEXT(int c) {   // what a WMO weather code means, in words
  if (c == 0) return "Clear"; if (c == 1) return "Mostly clear"; ...
}
...
// USER CODE END globals
```

The weather station keeps a lot here: the words for each weather code, its own fuller weather fetch (`wx_fetch()`, which asks Open-Meteo for every hour and the week, more than the designer's basic weather), and `wx_show()`, which puts it all on the screens.

The handlers

```
static void ui_next_clicked(lv_event_t * e) {
  lv_obj_t * obj = (lv_obj_t *) lv_event_get_target(e);
  lv_screen_load_anim(ui_hours, LV_SCR_LOAD_ANIM_MOVE_LEFT, 300, 0, false);
  // USER CODE BEGIN ui_next_clicked
  // USER CODE END ui_next_clicked
}
```

One function for each widget that does something. The designer wrote the line that goes to the Next 24 hours screen, because that's what **When it is pressed → Go to screen** says. Its USER CODE section is empty: the weather station needs nothing more there.

Wi-Fi, the time and the weather

`ui_net_begin()` joins Wi-Fi and sets the clock; `ui_weather()` fetches the weather now from Open-Meteo, and the nearest airport's own report; `ui_txt_town()`, `ui_txt_hm()` and their kind turn readings into text. The designer writes only the ones your design uses (see Wi-Fi, the time and the weather).

ui_update() and ui_create()

```
// shows what the board's inputs read (from loop(), five times a second)
void ui_update(void) {
  char b[64];
  snprintf(b, sizeof(b), "%s", ui_txt_town().c_str()); ui_text(ui_place, b);
  snprintf(b, sizeof(b), "%s", ui_txt_hm().c_str()); ui_text(ui_time, b);
  ui_wifi_icon(ui_wifi);
  ui_weather_pic(ui_wnow);
}
```

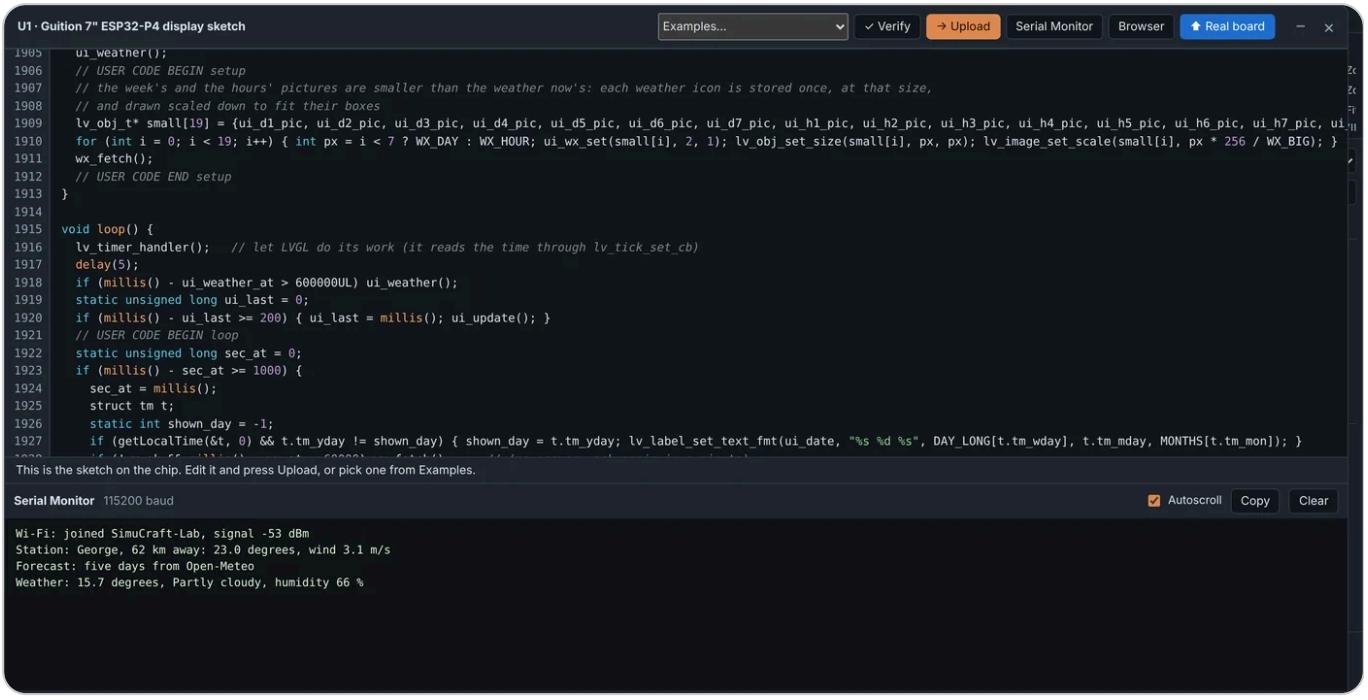
`ui_update()` is the designer showing the readings you picked under **On the board → It shows**: the town in `ui_place`, the time in `ui_time`, the Wi-Fi icon, the weather picture. Then `ui_create()`, usually the longest function in the sketch, builds every screen and widget with its position, size, font and colour.

setup() and loop()

```
void setup() {
  Serial.begin(115200);
  jc_begin();          // the screen, its touch chip, and LVGL on them
  ui_create();
  ui_ws_attach(ui_wifi);   // tap the Wi-Fi icon to choose a network
  lv_timer_handler();  // draw the screen now, before Wi-Fi (which can take a few
seconds)
  ui_net_begin();      // Wi-Fi and the time, then the weather
  ui_weather();
  // USER CODE BEGIN setup
  wx_fetch();
  // USER CODE END setup
}

void loop() {
  lv_timer_handler();  // let LVGL do its work
  delay(5);
  if (millis() - ui_weather_at > 600000UL) ui_weather();
  static unsigned long ui_last = 0;
  if (millis() - ui_last >= 200) { ui_last = millis(); ui_update(); }
  // USER CODE BEGIN loop
  ...
  // USER CODE END loop
}
```

The designer's part comes first in both, so by the time your setup code runs, the screen exists, Wi-Fi is up and the weather has been fetched once. In `loop()`, LVGL gets its turn, the weather is fetched again every ten minutes, the readings are shown five times a second, and then your code runs.



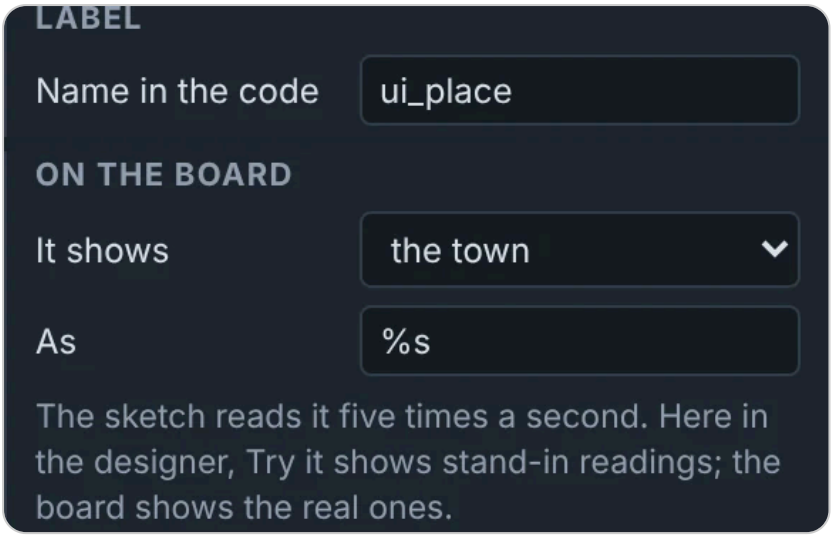
The board's Code window: this is where you write your part. Verify checks it; the Serial Monitor shows what it prints.

3. What the designer does without code

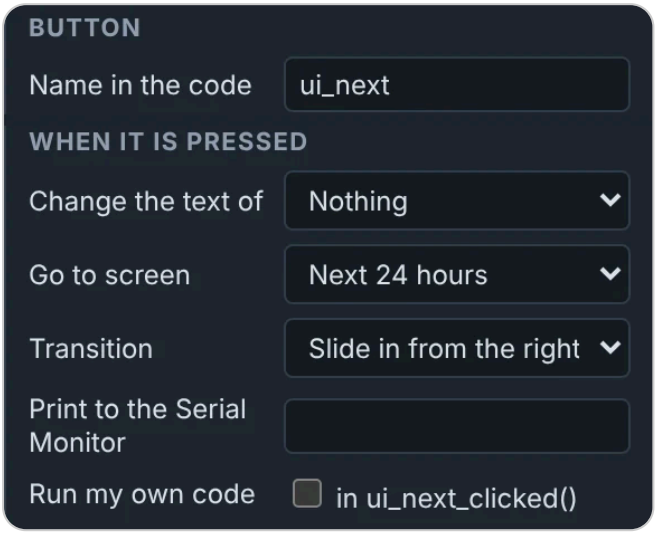
Before writing a line, see whether the designer will write it for you. A good deal of what screens do is already in its properties panel:

You want	In the designer	It writes
A button that opens another screen	Select it → **When it is pressed** → **Go to screen**, and a transition	`lv_screen_load_anim(…)` in its handler
A slider's number on a label	**When it changes** → **Show the value on** → the label, and its format (`%d %%`)	`lv_label_set_text_fmt(…, value)`
A switch that changes a label's words	**When it changes** → **Change the text of**, with the on and off words	`lv_label_set_text(…, on ? … : …)`
To see presses in the Serial Monitor	**Print to the Serial Monitor**, with the text	`Serial.printf(…)`
A label with the time, the date, the town, the temperature, the Wi-Fi signal	Select the label → **On the board** → **It shows**, then **As** for the words around it	a line in `ui_update()`, and the Wi-Fi and weather code it needs
The weather as a picture	A Picture → **It shows** → the weather, as a picture	the twelve weather icons in `ui_assets.h`, and `ui_weather_pic()`
Wi-Fi chosen on the screen, with a keyboard	**+ Wi-Fi screens** under the screen tabs	the Wi-Fi screens and `ui_ws_…` functions
The screen dimming when untouched, and at night	The screen → **Brightness** → **The designer handles it**	`ui_bl_begin()`, `ui_bl_loop()` (see Brightness)
A slider that sets the brightness	The slider → **It sets the brightness**	the slider's handler, kept in flash
A label greyed with "Off" when something is off	**When it is off** → the text and colour	`ui_alarm_set_off(bool off, …)` for you to call

You want	In the designer	It writes
Something hidden until needed	**Hidden at the start**	`lv_obj_add_flag(…, LV_OBJ_FLAG_HIDDEN)` in `ui_create()`



The town label: **On the board → It shows → the town**. No code: the designer adds a line to `ui_update()`.



The Next 24 h button: **Go to screen** does the work. Tick **Run my own code** when you want a USER CODE section in its handler as well.

Each of these can be combined with your own code: the designer's lines run first, then yours.

4. Your five places

There are only a few places your code can go, and each has a job:

Section	Runs	Put here	Don't put here
`globals`	never by itself: it holds things	your variables, constants, `#include`s for extra libraries, your own functions	statements outside a function
`setup`	once, after the screen, Wi-Fi and the first weather	starting your sensors, reading saved settings, filling the screen the first time	long waits: the screen shows nothing new until setup ends
`loop`	over and over, about every 5 ms	checking the time, reading sensors now and then, updating labels when something changed	`delay()` of more than a few milliseconds; work done every pass that only needs doing every second
`ui_…_clicked`	when that button is released	what the press does: start, stop, count, save	slow work such as a web request (start it from `loop`: see 7)
`ui_…_changed`	when that slider, arc, switch, list or roller changes	using the new `value`, `on` or `choice`	the same: keep it quick

A handler's section only exists if the design gives it one: a button with nothing to do has no handler. To get one, tick **Run my own code** in its **When it is pressed** (or **When it changes**).

The rules the designer keeps

- **Your code is matched by its section's name.** Rename a widget in the designer and its handler's name changes, from `ui_go_clicked` to `ui_start_clicked`; the designer carries your code across, and renames `ui_go` to `ui_start` in your code too.
- **Delete a widget, or untick Run my own code, and your code is still kept.** The next time the designer writes the sketch, it moves the code from the lost handler to the end of your globals, between `#if 0` and `#endif` so it isn't compiled, with a note saying where it came from. Move it where it belongs, or delete it.
- **Never edit outside the markers.** A change to `ui_create()` lasts until your next change in the designer. If you want a widget to look different, change it in the designer. If you must do something the designer can't, do it from your `setup` section, after `ui_create()` has run (see 6).

- **Keep the marker lines.** Delete a `// USER CODE END` line and the designer can't tell where your code stops.

5. Names you can use

Your widgets and screens

Every widget is an `lv_obj_t *` with its **Name in the code**: `ui_place`, `ui_next`. Screens too: the Weather screen is `ui_main`, Next 24 hours is `ui_hours` (set under **This screen → Name in the code**). Give widgets you will use in code names that say what they are; the designer's own (`ui_label_7`) are hard to read a month later.

A button's words are a label inside it, made by `ui_create()`; reach them as the button's first child: `lv_label_set_text(lv_obj_get_child(ui_next, 0), "Tomorrow");`

In a handler

Name	In	What it is
`obj`	every handler	the widget that was pressed or changed
`value`	a slider's or arc's	its new value, between the Min and Max you set
`on`	a switch's or check box's	`true` when it is on
`choice`	a drop-down list's or roller's	the words of what was picked; `lv_dropdown_get_selected(obj)` gives its number

The designer's helpers

Name	What it does
`ui_text(label, "…")`	sets a label's text only if it changed (no needless redraw; on e-paper, no needless refresh)
`ui_t`, `ui_tmax`, `ui_tmin`, `ui_rain`, `ui_wind`, `ui_wdir`, `ui_uv`	the weather now, as numbers (`NAN` until it arrives)
`ui_wcode`, `ui_wday`	the weather code (WMO) and whether it is day
`ui_town`, `ui_lat`, `ui_lon`	the town and its position, from the designer's Wi-Fi, time and weather
`ui_weather_at`	when the weather last arrived (`millis()`); set it to 0 to fetch again now
`ui_txt_hm()`, `ui_txt_date()`, `ui_txt_temp()`…	a reading as text, ready for a label
`ui_wx_set(picture, code, day)`	shows the right weather icon in a picture
`ui_bl_…`	the brightness (see 9)

Which helpers a sketch has depends on its design: `ui_wx_set` only if it shows the weather as a picture, for example. If you call one that isn't there, the sketch won't build; add what needs it in the designer, or write your own.

Why your code can use names declared further down. C++ wants everything declared before it is used. The designer puts the widgets' names and its variables (as `extern`) above your globals, and when the sketch goes to a board SimuCraft moves any other declarations your code needs above it as well. If you take the sketch to the Arduino IDE and it complains that something "was not declared in this scope", add a declaration of it at the top of your globals.

6. Changing the screen from code

Most of what a sketch does to its screen is one of these, and every one is a single LVGL call:

```
// text
lv_label_set_text(ui_place, "Knysna");
ui_text(ui_place, "Knysna");                       // the same, only if it changed
lv_label_set_text_fmt(ui_count, "%d visitors", n); // whole numbers: %d, %u, %ld; and %s

// numbers on bars, sliders, arcs
lv_bar_set_value(ui_level, 72, LV_ANIM_ON);
lv_slider_set_value(ui_volume, 40, LV_ANIM_OFF);
lv_arc_set_value(ui_dial, 18);

// on or off
lv_obj_add_state(ui_heat, LV_STATE_CHECKED);       // a switch on
lv_obj_remove_state(ui_heat, LV_STATE_CHECKED);    // and off
if (lv_obj_has_state(ui_heat, LV_STATE_CHECKED)) { ... }

// show and hide
lv_obj_add_flag(ui_warning, LV_OBJ_FLAG_HIDDEN);
lv_obj_remove_flag(ui_warning, LV_OBJ_FLAG_HIDDEN);

// colour
lv_obj_set_style_text_color(ui_temp, lv_color_hex(0xef4444), 0);   // red words
lv_obj_set_style_bg_color(ui_card, lv_color_hex(0x1e293b), 0);     // a panel's
background

// pictures (from ui_assets.h)
lv_image_set_src(ui_icon, &img_fan_on);

// another screen
lv_screen_load_anim(ui_hours, LV_SCR_LOAD_ANIM_MOVE_LEFT, 300, 0, false);
```

Decimals: never %f

The LVGL built for the boards has no floating-point printing, to save space. A `%f` or `%.1f` in `lv_label_set_text_fmt` prints just the letter **f**: that is where a weather screen's "f mm" came from. Make the text first, with the C library's `snprintf`, which does know decimals, then set it:

```
char b[16];
snprintf(b, sizeof(b), "%.1f mm", ui_rain);   // snprintf, not lv_label_set_text_fmt
lv_label_set_text(ui_rain_lbl, b);
// or with Arduino's String: lv_label_set_text(ui_rain_lbl, (String(ui_rain, 1) + "
mm").c_str());
```

SimuCraft's simulator prints **f** just as the board would, and **Verify** warns you when it finds a `%f` going to LVGL.

Only when it changed

Every time you set a label, LVGL redraws it, even to the same words. In `loop()`, which runs two hundred times a second, that is wasted work, and on an e-paper screen it's a visible flash. Keep the last value and set the label only when it differs, or use `ui_text()`, which compares for you:

```
static int shown = -1;
if (count != shown) { shown = count; lv_label_set_text_fmt(ui_count, "%d", count); }
```

Changing what the designer built

Everything `ui_create()` made, your `setup` section can change, because it runs after. The weather station does it for its small weather icons: the designer stores each icon once, at the big size, and the station's setup code shrinks the week's and the hours' copies with `lv_image_set_scale`. Keep these changes few, and in one place; if you find yourself restyling a widget from code, it is usually quicker to restyle it in the designer.

7. Doing things over time

The screen only moves while `lv_timer_handler()` runs, and that happens at the top of `loop()`. Anything that holds up `loop()` freezes the screen: touches aren't felt, animations stop, the clock stops ticking. So:

- **No long `delay()`.** A `delay(1000)` in your loop section makes the screen respond once a second.
- **Time things with `millis()`.** Keep when you last did something, and do it again when enough time has passed. This is how the designer does its own loop, and how the weather station does its:

```
// USER CODE BEGIN loop
static unsigned long sec_at = 0;
if (millis() - sec_at >= 1000) {      // once a second
  sec_at = millis();
  struct tm t;
  static int shown_day = -1;
  if (getLocalTime(&t, 0) && t.tm_yday != shown_day) {   // the date, once a day
    shown_day = t.tm_yday;
    lv_label_set_text_fmt(ui_date, "%s %d %s", DAY_LONG[t.tm_wday], t.tm_mday,
MONTHS[t.tm_mon]);
  }
  if (millis() - wx_at > 15UL * 60 * 1000) wx_fetch();   // the weather, every 15 minutes
}
// USER CODE END loop
```

- **Slow work in small pieces, or with a short timeout.** A web request can take seconds. Give it a timeout (`http.setTimeout(6000)`), do it every few minutes rather than every pass, and never from a handler: a handler that waits for the internet holds the button down.
- **A press that starts slow work sets a flag.** The handler says "fetch now"; `loop()` does it:

```
// globals
bool want_fetch = false;
// the Refresh button's handler
want_fetch = true;
// loop
if (want_fetch) { want_fetch = false; wx_fetch(); }
```

- **LVGL timers** are the other way: `lv_timer_create(my_fn, 1000, NULL)` calls `my_fn(lv_timer_t * t)` every second from inside `lv_timer_handler()`. They suit screen-only work, such as blinking a label. The designer uses one to watch Wi-Fi.
- **Don't touch widgets from another task.** LVGL isn't thread-safe. If you start a FreeRTOS task (the internet radio does, for its audio), have it set variables, and let `loop()` put them on the screen.

8. Wi-Fi, the time and the weather

Select the screen itself (click an empty part of it) and look at **Wi-Fi, time and weather**. The designer writes this part of the sketch only if something in the design needs it: a label showing the time needs Wi-Fi and the clock; a label showing the temperature also needs the weather.

- **Wi-Fi network and Password**: what the board joins. When you upload, SimuCraft asks for your own and puts them in; in the simulator, boards join the lab network. With **Wi-Fi screens** added, the board also keeps networks chosen on its own screen, and tries those first.
- **Units**, **Town**, **Latitude**, **Longitude** and **Time zone**: what the weather and the clock are for. **Find the town by itself** looks the town up from the board's internet address instead.

When you need more than the designer fetches, fetch it yourself, as the weather station does with `wx_fetch()`: the designer's `ui_weather()` asks for the weather now and today; the station's own code asks for every hour and the week. Use the designer's `ui_lat` and `ui_lon` in your request, so the town set in the designer is the one your code asks about.

> On the board, Wi-Fi and HTTPS work through the Wi-Fi chip beside the processor (the ESP32-C6 on the Guitions and the Tab5). SimuCraft brings that chip's software up to date the first time the sketch runs; the designer writes it in for you as `ui_c6_update()`.

9. Brightness and the off look

Two jobs that used to need a page of code each are now designer settings.

Brightness

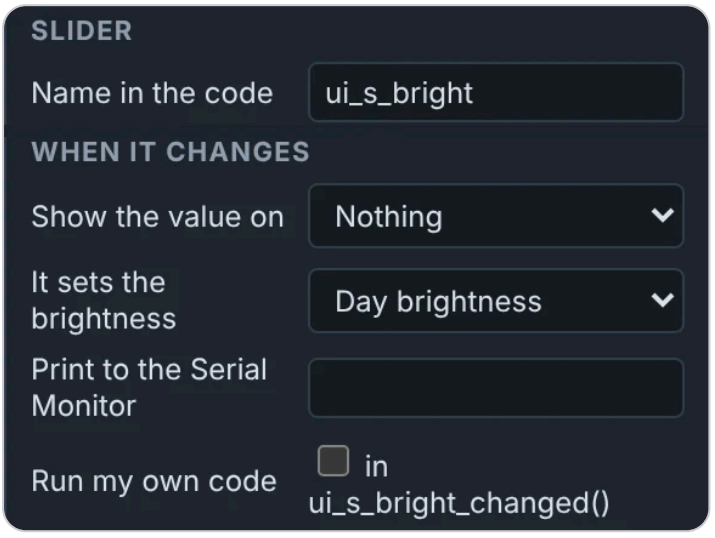
Select the screen, find **Brightness** and tick **The designer handles it**. Then choose the daytime brightness, how long until it dims when nobody touches it, whether it is dimmer at night and

between which hours, the night brightness, a floor it never goes below, how long a touch at night brightens it, and whether the first touch on a dimmed screen only wakes it. **Show it here** previews the level on the designer's screen.



The bedside clock's brightness, set in the designer. The levels are on a log scale, as the eye sees them.

The designer writes `ui_bl_begin()` into setup and `ui_bl_loop()` into loop, and keeps the settings in flash, so a slider on the board that changes them is remembered through a power cut. To let people set it on the board, put a slider (or a roller for the times) on a settings screen and choose what it sets under **It sets the brightness**: no code.



The clock's Setup screen: this slider sets the daytime brightness.

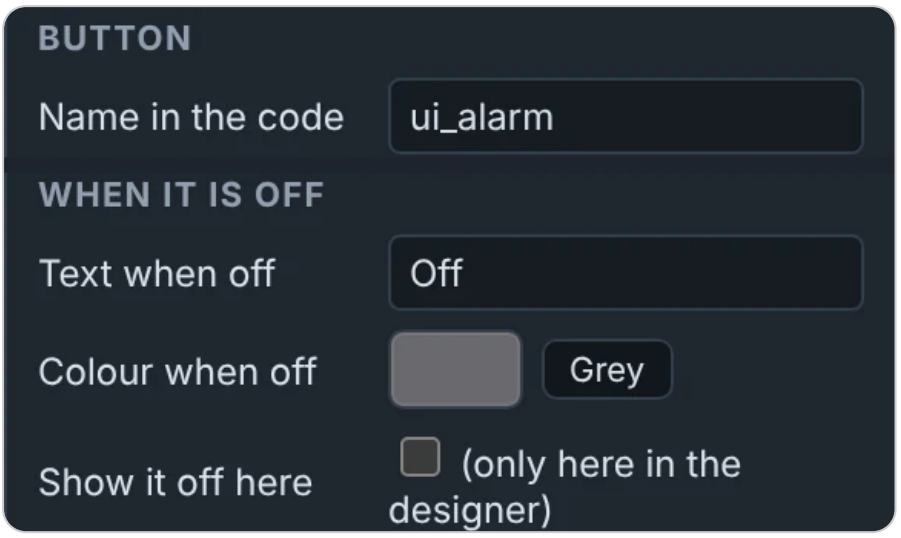
Your code can still take charge when it must: set `ui_bl_force` to a level (0 to 4095) and the screen holds it; set it back to `-1` to hand back to the designer. The clock does this for the alarm:

```
ui_bl_force = ringing ? 4095 : sunrise ? map(20 - before, 0, 20, 64, 4095) : -1;
```

The off look

A label or button that says something is switched off, such as the clock's alarm time when the alarm is off, can be given an off look: select it, and under **When it is off** set the text (for example **Off**) and its colour. **Show it off here** previews it. The designer writes a function named after the widget:

```
ui_alarm_set_off(!alarm_on, "06:30");   // off: "Off" in grey; on: "06:30" in its own colour
```



The clock's alarm: the words and colour for when it is off are design, not code.

10. The round trip

Working on a designed sketch goes round the same loop:

1. **Design.** Change the screen in the designer. The simulated board follows within a second: the button at the top says ✓ **Board up to date**. (Until the board runs a designer's sketch, it says **Send to the board**.)
2. **Try it** in the designer to press buttons and drag sliders on the design itself, or use the board in the circuit, where your code runs too.
3. **Code.** Write your part in the board's Code window, between the markers. **</> Code** in the designer shows the whole sketch as it will be written, read-only, which is handy for finding names.
4. **Verify** in the Code window. It catches what the compiler would, and the things that work in the simulator but not on a board, such as `%f`.
5. **Real board** builds the sketch and puts it on the board, with `ui_assets.h` and the Wi-Fi chip's file.

Going the other way: **⬇ Save for Arduino IDE** saves `screen_sketch.ino` and, if the design has pictures or fonts, `ui_assets.h`. Put both in one folder named `screen_sketch`. If you then edit in

the Arduino IDE, paste the sketch back into SimuCraft's Code window before designing again: the designer picks up your sections from it.

And a sketch you wrote by hand, without the designer? Open **Design screen** and the designer reads back the screen that is showing. If you send its sketch to the board, it replaces yours, so it asks first.

11. Worked example: a Refresh button

Let's add a button to the weather station that fetches the weather now, and a label saying how old the weather is. It uses most of this lesson.

1. **Add the button.** In the designer, drag a **Button** onto the Weather screen beside Next 24 h. Set its text to **Refresh** and its **Name in the code** to `ui_refresh`. Under **When it is pressed**, tick **Run my own code**.
2. **Add the label.** A **Label** under it, named `ui_age`, text **–**, a small font.
3. **The flag, in globals.** In the Code window, find `// USER CODE BEGIN globals` and add:

```
bool want_fetch = false;   // set by Refresh, done in loop()
```

4. **The press.** The designer has made `ui_refresh_clicked` with its own section. Add:

```
  want_fetch = true;
  lv_label_set_text(lv_obj_get_child(ui_refresh, 0), "Fetching…");
```

5. **The work, in loop.** At the start of the loop section:

```
  if (want_fetch) {
    want_fetch = false;
    wx_fetch();                 // the station's own fetch: now, the hours, the week
    lv_label_set_text(lv_obj_get_child(ui_refresh, 0), "Refresh");
  }
```

6. **How old, once a minute.** Inside the loop section's once-a-second block, add:

```
    static long shown_min = -1;
    long mins = (millis() - wx_at) / 60000;
    if (mins != shown_min) {
      shown_min = mins;
      lv_label_set_text_fmt(ui_age, mins == 0 ? "Just now" : "%ld min ago", mins);
    }
```

   Whole minutes, so `%ld`: no decimals, no `%f`.
7. **Verify, then try it.** Press Refresh on the simulated board: the button says Fetching..., the screen fills, the age goes back to Just now.

8. **Change the design again**: move the button, make it green. The designer rewrites the sketch; open the Code window and your four additions are where you left them.

12. Pitfalls

My change disappeared

It was outside the markers. Anything between `// USER CODE BEGIN` and `END` is kept; the rest is the designer's and is rewritten after every change. Do the change in the designer, or from your setup section.

A label shows "f" or "f mm"

A `%f` went to LVGL. Use `snprintf` (6).

The screen freezes or touches lag

Something holds up `loop()`: a `delay()`, a slow web request, a loop that waits for a sensor. Time it with `millis()`, and move slow work out of handlers (7).

"'ui_...' was not declared in this scope"

The widget has a different name in the design (check **Name in the code**), or it has been deleted. If it's one of your own functions, used before it is written, declare it at the top of your globals: `void wx_show();`

My handler's code ended up under #if 0

Its handler is gone: the widget was deleted or renamed outside the designer, or Run my own code was unticked. The code is safe in your globals; move it into the new handler and delete the `#if 0` and `#endif` lines.

The label flickers, or the e-paper flashes

It is set every pass of `loop()`. Set it only when its value changes, or use `ui_text()`.

It works in SimuCraft but not on the board

Run Verify, and read the Serial Monitor on the real board: most differences are Wi-Fi (the board needs a 2.4 GHz network), a slow web request that the lab network answers at once, or `%f`. The From simulation to the real board lesson goes through the rest.

Checklist

- Looks, positions, screens, what labels show: in the designer.
- Your code: only between the USER CODE markers.
- Variables and functions in globals; start-up in setup; repeated work in loop, timed with `millis()`.
- Handlers quick; slow work started with a flag and done in loop.
- No `%f` in LVGL calls: `snprintf` first.
- Labels set only when they change.
- Verify before every upload; watch the Serial Monitor on the board.

Next, see the rules at work in three complete examples: The clock radio, The weather station and The Home Assistant wall panel.

Revision history

Revision	Date	What changed
1.0	21 November 2026	First edition.