LESSON · EVERY BOARD

# From the simulator to the real board

Revision 1.1 · 21 November 2026

A sketch that runs in SimuCraft is nearly ready for a real board, but not quite: the board's compiler is stricter than the simulator, it needs every library on your PC, and its first build takes a while. This lesson explains what happens between pressing Upload and seeing your sketch run, what SimuCraft does for you on the way, and how to read the messages when something stops it. The examples are real mistakes, found while the Guition boards were tested.

## What you need

- **SimuCraft for Windows**, 0.6.5 or later. Steps 1, 4 and 6 also work in the web version; compiling for a real board needs the desktop app.
- Any board SimuCraft can upload to, and a USB cable that carries data, if you want to try the last steps for real. You can do most of the lesson with **Check it compiles** alone, with no board plugged in.
- About 45 minutes, plus one first build (see step 3).

## 1. Two ways to run a sketch

When you press **Upload** on a chip in SimuCraft, the sketch runs in the simulator: SimuCraft turns your C++ into JavaScript and runs it in the app, with the simulated pins, screen and Serial Monitor. That is quick, and it lets you try things without a board.

When you press **Check it compiles** or **Upload to the board** in the real-board window, SimuCraft hands the same sketch to **Arduino's own tools** (arduino-cli, the engine inside the Arduino IDE). They compile it with the board's real C++ compiler into machine code for that chip, and copy it over USB.

Almost everything you write behaves the same in both. The differences are the subject of this lesson:

- The real compiler needs every **library** the sketch uses installed on your PC (step 2).
- The real compiler builds the board's whole **core** once, which is why the first build is slow (step 3).
- The real compiler is strict about **order**: a name has to be declared above the place it's used (step 4). The simulator doesn't mind, so a sketch can run in SimuCraft and still be refused by the board's compiler.

## 2. #include and libraries

A line like `#include <ArduinoJson.h>` pastes in a **header**: the list of things a library offers (its classes, functions and settings). The header belongs to a **library**, a folder of code that the compiler builds along with your sketch. There are three kinds:

- **Built into the board's package**: `Wire.h`, `SPI.h`, `WiFi.h`, `Preferences.h` on the ESP32 boards. They arrive with the board's tools.
- **From Arduino's Library Manager**: `ArduinoJson.h`, `lvgl.h`, GFX Library for Arduino.
- **From the author's GitHub**, for libraries not in the Library Manager, such as ESP32-audioI2S (`Audio.h`) for internet radio.

You don't install any of these yourself. When you press **Check it compiles**, SimuCraft reads the sketch's `#include` lines and installs each library it knows, in the version our examples are tested with (LVGL 9.2.2, for example: the newest LVGL needs a different settings file). If the compiler still reports a missing header, SimuCraft looks it up in the Library Manager, installs it and builds again. The lines in the window say what it is doing: *Installing the ArduinoJson library...*, *Fetching the ESP32-audioI2S-master library from its author's GitHub...*

> **<Angle> or "quotes"?** `#include <Name.h>` looks in the libraries; `#include "Name.h"` looks in the sketch's own folder first, then the libraries. A designed screen's pictures and fonts are in `ui_assets.h`, a file in the sketch's folder, so that one uses quotes.

## 3. Why the first build is slow

The first time you compile for a board, SimuCraft and Arduino's tools do a lot of work you only see once:

1. **They download the board's tools**: the compiler and the board's package. For the ESP32 boards that is several hundred megabytes; for an Arduino Uno much less.
2. **They install the libraries** the sketch needs (step 2).
3. **They compile the board's whole core and every library**: for an ESP32-P4 with LVGL that is thousands of files. This is the long part: typically 5 to 10 minutes on an ESP32, a minute or two on an Uno.

Everything compiled is kept. The next build compiles only your sketch, usually in under a minute. The progress line keeps changing while it works: if it moves, it isn't stuck.

A build is slow again when something it kept no longer fits:

- a board you haven't compiled for before;
- a change to the board's settings (for the Guition boards, the **Chip Variant**, which SimuCraft sets for you);

- a library updated or newly installed;
- the build folder deleted (in `%APPDATA%\SimuCraft\arduino\builds`), which some fixes ask you to do.

## 4. Declare it before you use it

C++ reads a sketch from the top down. When it meets a name, it must already know what that name is. So a variable has to be **declared** above the first function that uses it:

```
void setup() {
  count = 0;          // the compiler stops here: what is count?
}
void loop() { }
int count;            // declared too late
```

The fix is to move `int count;` above `setup()`. The real compiler says `'count' was not declared in this scope`. The simulator runs JavaScript, which doesn't mind the order, so this sketch runs in SimuCraft.

That's why SimuCraft's ✔ **Verify** now checks the order the way C++ does. Press it, and it names the line:

> *Line 2: count is used here but only declared on line 5. It runs in SimuCraft, but Arduino's compiler for the real board stops: move the declaration on line 5 above the function that uses it.*

**Functions are different.** You can call a function that's written further down: Arduino's tools write a one-line **prototype** for each function near the top of the sketch for you. They do that for functions only, not for variables, so variables are the ones to watch.

## 5. What SimuCraft puts in order for you

Sketches made in the **screen designer** have sections the designer writes and sections for your own code (between the `USER CODE BEGIN` and `END` lines). Your code often uses the designer's names: a label such as `ui_set`, or the town's `ui_lat` for the weather. When a designed sketch goes to the board, SimuCraft puts three things in order, so you can write your code where the sections are:

- **The screen's widgets** (`lv_obj_t * ui_...`) are declared above your code.
- **The designer's own variables** (Wi-Fi, time and weather: `ui_town`, `ui_lat`, ...) are declared above your code with `extern`, a line that says "this exists, and is defined further down".
- **Your #include lines** move up to the top with the sketch's own, stopping at a `#define` or `#if` that might set a library up.

Your sketch in the editor stays as you wrote it: the changes are made to the copy that goes to Arduino's tools. Verify leaves these out of its warnings, since SimuCraft deals with them.

> **Keep #include lines short.** A comment on an `#include` line longer than about 465 characters makes the compiler number every line after it one short. Error messages then point at the line above the real one, and a missing library can't be found (story 3 below). SimuCraft shortens such lines on the way to the board, but in your own sketches put long notes on lines of their own.

## 6. Reading a compile error

When a build fails, the real-board window shows the first error in red, with the line it's on, and the full output under **Details** (the **Copy** button copies it, for a forum post or for us).

- **Fix the first error first.** One mistake often causes a chain of others: a missing `}` can produce twenty messages. The count in brackets, *(and 14 more)*, often drops to nothing once the first is fixed.
- `'x' was not declared in this scope`: a name used above its declaration (step 4), or a spelling mistake, or a library's header missing.
- `Name.h: No such file or directory`: a library isn't installed. SimuCraft normally installs it and builds again; if it can't find one that provides the header, it says so.
- `multiple definition of 'x'`: the same thing defined twice, often in two files of one sketch.
- ***Sketch too big* or *text section exceeds available space***: the program doesn't fit the board's flash. Pictures and fonts are usually the reason: use fewer or smaller ones.
- ***The sketch needs more memory (RAM)***: big arrays. Make them smaller, or allocate them with `malloc()` so they go into the board's PSRAM where it has some.

## 7. Four real mistakes

The first three ran happily in SimuCraft and stopped on the way to a real Guition ESP32-P4 board; the fourth compiled and ran, but showed the wrong thing. They're fixed now, and each fix has a test so it can't come back.

### The wall panel: `'ui_set' was not declared in this scope`

The smart-home wall panel's own code set the thermostat's label, `ui_set`, in a function in the USER CODE section. The designer declared its widgets below that section, so the compiler met `ui_set` before it knew what it was: 15 errors, one for each widget used. Now the widgets are declared first (step 5).

### The weather station: `'ui_lat' was not declared in this scope`

The same thing with the designer's weather variables: the town's latitude and longitude, used by the code that asks Open-Meteo for the forecast, were defined further down. 103 errors from one cause. Now they're declared with `extern` above your code.

### The clock radio: `Audio.h: No such file or directory`

This one took longer. The Audio library was installed, but the build still couldn't find it. The reason: the line `#include "ui_assets.h"` had a comment listing every picture and font, over 500 characters long. After such a line the compiler numbers the following lines one short. It reported the missing header on the line above the real one, `#include <Preferences.h>`, so Arduino's tools looked for the Preferences library (which the board already has) instead of the Audio library. The list now goes on comment lines of its own.

### The weather screen: "f mm" where the rain should be

On a real 7" board's weather screen the rain read **f mm**. The sketch set the label with `lv_label_set_text_fmt(ui_rain, "%.1f mm", rain)`, which is right on a PC, but the LVGL built for the boards leaves out floating-point printing to save space, and prints the letter instead. The examples now make the text with `snprintf`, which does know decimals, then set it; SimuCraft's simulator prints the f just as the board does, and Verify warns when a `%f` goes to LVGL.

## If something goes wrong

More fixes for uploads, ports and drivers are in Uploading over USB.

- **It runs in SimuCraft but won't compile.** Press ✓ Verify first: it names any order problem. Then read the first error in the real-board window (step 6).
- **A library is installed but its header is "No such file".** Delete the sketch's build folder (`%APPDATA%\SimuCraft\arduino\builds\SimuCraft_…`) and build again: the tools keep a note of where headers were found, and an old note can hide a new library.
- **The first build seems to take for ever.** Watch the progress line: it changes while files compile. On an ESP32 with LVGL, 10 minutes is normal the first time.
- **Cancel** stops Arduino's tools straight away if you started the wrong build.

## Things to try

- Open any Arduino example, move a global variable below `loop()`, and press ✓ Verify. Then move it back.
- Add `#include <ArduinoJson.h>` to a sketch, press **Check it compiles**, and watch SimuCraft install the library.

- Time a first build and a second build for the same board. Change the board's Chip Variant (a Guition board) and time it again.
- Break a sketch on purpose with a missing `}`, and see how many errors one mistake makes.

## Revision history

| Revision | Date | What changed |
|---|---|---|
| 1.0 | 16 November 2026 | First edition. |
| 1.1 | 21 November 2026 | A fourth real mistake: %f printing "f" on the board. |