Ubuntu (Linux) — Build & Run Guide for `qt_view.cpp`

This guide shows how to compile and run the **Qt GUI** for your project on Ubuntu (22.04/24.04). It assumes you already have the shared core headers and the latest **MOC-free** `qt_view.cpp` (no `Q_OBJECT`, no `#include "qt_view.moc"`).

---

0) Project layout

Place these files in one folder (e.g., `~/scripted`):

```
scripted/
├── scripted_core.hpp
├── frontend_contract.hpp
├── presenter.hpp
├── qt_view.cpp
├── scripted.cpp              # (CLI, optional)
└── files/                    # your data (config.json, x00001.txt, ...)
```

> The GUI and CLI both expect a `files/` directory in the working directory.

---

1) Install dependencies

Ubuntu 24.04 / 22.04

```
sudo apt update
sudo apt install -y build-essential pkg-config qt6-base-dev
```

> If you later get Qt plugin errors (xcb), see the **Troubleshooting** section.

---

2) Build (one-liner)

Use `pkg-config` to locate Qt6:

```
c++ -std=c++23 -O2 qt_view.cpp -o scripted-gui `pkg-config --cflags --libs Qt6Widgets`
```

- If your compiler doesn't accept `-std=c++23`, use `-std=c++20` instead:

  ```
  c++ -std=c++20 -O2 qt_view.cpp -o scripted-gui `pkg-config --cflags --libs Qt6Widgets`
  ```

---

3) Run

From the same folder (so `files/` is found):

```
./scripted-gui
```

---

4) Optional: build the CLI too

```
c++ -std=c++23 -O2 scripted.cpp -o scripted
./scripted
```

---

5) CMake build (recommended for repeat builds)

Create `CMakeLists.txt`:

```
cmake_minimum_required(VERSION 3.22)
project(scripted_gui_qt LANGUAGES CXX)

set(CMAKE_CXX_STANDARD 23)
set(CMAKE_CXX_STANDARD_REQUIRED ON)

find_package(Qt6 REQUIRED COMPONENTS Widgets)

add_executable(scripted-gui qt_view.cpp)
target_include_directories(scripted-gui PRIVATE ${CMAKE_SOURCE_DIR})
target_link_libraries(scripted-gui PRIVATE Qt6::Widgets)
```

Build:

```
mkdir -p build && cd build
cmake ..
cmake --build . -j
./scripted-gui
```

> If your compiler doesn't support C++23, add `set(CMAKE_CXX_STANDARD 20)` instead.

---

6) Common warning & quick fix

Warning:

```
QStandardPaths: wrong permissions on runtime directory /run/user/1000/, 0755 instead of 0700
```

Fix (Linux/WSL):

```
sudo chmod 700 /run/user/1000
ls -ld /run/user/1000   # should show: drwx------
```

---

7) Troubleshooting

A) Qt “xcb” plugin errors (cannot load platform plugin)

Install xcb-related libraries:

```
sudo apt install -y libxcb1 libxkbcommon-x11-0 libxcb-icccm4 libxcb-image0 \
  libxcb-keysyms1 libxcb-render0 libxcb-render-util0 libxcb-shape0 \
  libxcb-xfixes0 libxcb-randr0 libxcb-xinerama0 libxcb-glx0
```

Then run:

```
./scripted-gui
```

(For deeper diagnostics: `export QT_DEBUG_PLUGINS=1` before running.)

B) `pkg-config --cflags --libs Qt6Widgets` prints nothing / errors

Install Qt6 and pkg-config:

```
sudo apt install -y qt6-base-dev pkg-config
```

C) Compiler doesn’t recognize `-std=c++23`

Use `-std=c++20`:

```
c++ -std=c++20 -O2 qt_view.cpp -o scripted-gui `pkg-config --cflags --libs Qt6Widgets`
```

D) App can’t find `files/`

Run the app from the directory that contains `files/`, or pass an absolute working dir. The program expects to read/write in `./files`.

---

8) Notes about MOC (Qt’s Meta-Object Compiler)

- The **latest** `qt_view.cpp` is **MOC-free** (no `Q_OBJECT`, no `#include "qt_view.moc"`).

- If you ever reintroduce `Q_OBJECT` or custom signals/slots, you must either:
    - Enable CMake's `CMAKE_AUTOMOC`, or
    - Generate MOC manually before compiling:

      ```
      /usr/lib/qt6/libexec/moc qt_view.cpp -o qt_view.moc
      c++ -std=c++23 -O2 qt_view.cpp -o scripted-gui `pkg-config --cflags --libs Qt6Widgets`
      ```

---

9) Quick functional checklist

1. Launch `./scripted-gui` → status shows **Ready**.
2. **Preload** → loads contexts from `files/`.
3. In the combo, type `x00001` → **Switch** → rows appear.
4. Type in **Filter** → table narrows live.
5. **Insert/Update**: set Reg/Addr/Value → click button → row updates.
6. **Delete**: select row → **Delete**.
7. **Save**: writes back to `files/xNNNNN.txt`.
8. **Resolve**: produces `files/out/xNNNNN.resolved.txt`.
9. **Export JSON**: produces `files/out/xNNNNN.json`.
10. Cross-context references resolve correctly.