Umicom Studio IDE — Deep Research & Build Guide (Windows, PowerShell, Source-Only)

> **Author**: Umicom Foundation — Engineering Team
> **Scope**: Windows-first bring-up of Umicom Studio IDE (GTK4) with UAEngine + LLM backends (llama.cpp / Ollama / OpenAI), from-source dependency strategy, GitHub org workflow, and PowerShell-only commands.
> **Principles**: Source-first (no MSYS2/vcpkg for releases), fork under **umicom-foundation**, submodule every third-party, GCC as default compiler (Clang optional), TinyCC available but off by default, consistent MIT headers + clang-format.

---

0) Quick Status & Objectives

Current pain points (from repo + files you shared): - GTK4/GtkSourceView dev packages missing → configure/link errors - CMake targets partially miswired (e.g. curl linked to a non-existent target) → link errors - LLM backends: OpenAI/Ollama stubs → need real HTTP calls; llama.cpp embed needs vendoring + build - IDE shell incomplete: `app.c` / `window.c` basic scaffolding; SourceView integration missing - Inconsistent tree, mixed build scripts, missing headers/version glue

Bring-up objective (Windows): 1) Create/fork repos under **umicom-foundation** and clone under `C:\dev`
2) Build GTK4 & GtkSourceView from source (Visual Studio toolchain)
3) Build libcurl with native Windows Schannel (no OpenSSL req.)
4) Vendor **llama.cpp** and implement **OpenAI** & **Ollama** HTTP backends
5) Wire **GtkSourceView** editor, output console, and basic menu/actions
6) Provide **compiler switcher**: TCC (quick-run), GCC/Clang (full)
7) Normalize repo layout, add scripts, formatters, and MIT headers
8) Ship weekly **Windows zip** (GTK + IDE + tools) as release artifact

---

1) Canonical Repo Layout (proposed)

```
C:/dev/umicom-studio-ide/
├─ ide/                          # GUI app (GTK4) — app.c, window.c,
editor.c, ui assets
│  ├─ include/
│  ├─ src/
│  │  ├─ app.c
│  │  ├─ window.c
│  │  ├─ editor.c
│  │  ├─ console.c              # run/build output pane
│  │  ├─ ai_panel.c             # chat/FIM panel (later)
│  │  └─ studio_codestral_fim.c  # FIM demo/utility (Ollama or llama.cpp)
│  └─ data/
│     ├─ ustudio.gresource.xml
│     ├─ ui/main.ui              # GtkBuilder XML (menus, layout)
│     ├─ ui/settings.ui
│     ├─ icons/*.svg
│     └─ css/app.css
├─ uaengine/                     # AuthorEngine reusable core (no CLI main in
this lib)
│  ├─ include/ueng/
│  │  ├─ common.h  config.h  llm.h  fs.h  serve.h  version.h
│  └─ src/
│     ├─ common.c  config.c  fs.c  serve.c
│     ├─ llm_openai.c  llm_ollama.c  llm_llama.c
│     └─ ...
├─ uaengine_cli/                 # Optional standalone `uaengine` binary
│  └─ src/main.c
├─ third_party/
│  ├─ llama.cpp/                 # submodule fork (ggml-org/llama.cpp)
│  ├─ tcc/                       # submodule fork (TinyCC)
│  ├─ glad/                      # optional for GL examples
│  └─ ... other vendored libs
├─ scripts/
│  ├─ bootstrap-windows.ps1      # end-to-end tool & env verifier
│  ├─ env-gtk.ps1                # exports PATH/PKG_CONFIG_PATH for GTK
│  ├─ build-curl.ps1             # builds libcurl (Schannel)
│  ├─ build-gtk.ps1              # gvsbuild automation wrapper
│  ├─ build-ide.ps1              # CMake configure+build
│  └─ doctor.ps1                 # smoke tests (GTK window, UAEngine
selftest)
├─ cmake/
│  ├─ FindGtkSourceView.cmake    # if we need a custom finder
│  └─ toolchains/
└─ CMakeLists.txt                # orchestrates subdirs
```

Notes - Keep each C/HEADER file prefixed with the **MIT banner** you provided.
- `.clang-format` at repo root; run `clang-format -i` in CI.
- Generated sources (e.g. `ustudio-resources.c`) stay in CMake build dir.

---

2) PowerShell-Only — Org, Repos, Forks, Submodules

Requires: Git + GitHub CLI (`gh`) installed, and you're authenticated to **umicom-foundation**.

```
# 2.1 Authenticate GitHub CLI (one-time)
winget install --id GitHub.cli -e  # or: choco install gh
gh auth login --hostname github.com --git-protocol https --web

# 2.2 Ensure C:\dev exists
New-Item -ItemType Directory -Force -Path C:\dev | Out-Null
```

```
Set-Location C:\dev

# 2.3 Clone primary IDE repo
# (If it already exists, skip this and just `cd` into it.)
git clone https://github.com/umicom-foundation/umicom-studio-ide.git
Set-Location .\umicom-studio-ide

# 2.4 Create Umicom-GTK (separate repo) under org and clone
# (skip if already created)
Set-Location C:\dev
gh repo create umicom-foundation/umicom-gtk --public --description "GTK patches & helpers for Umicom" --clone

# 2.5 Fork third-party dependencies into org + clone locally
# llama.cpp → for local LLM inference
gh repo fork ggml-org/llama.cpp --org umicom-foundation --clone
# TinyCC → fast C compile & -run
gh repo fork TinyCC/tinycc       --org umicom-foundation --clone
# glad → optional GL loader for graphics samples
gh repo fork Dav1dde/glad        --org umicom-foundation --clone
# Ollama → local model server (API integration in IDE)
gh repo fork ollama/ollama       --org umicom-foundation --clone

# 2.6 Add submodules to IDE repo
Set-Location C:\dev\umicom-studio-ide
# Place submodules under third_party/
git submodule add https://github.com/umicom-foundation/llama.cpp third_party/llama.cpp
git submodule add https://github.com/umicom-foundation/tinycc third_party/tcc
git submodule add https://github.com/umicom-foundation/glad third_party/glad
# Optional: submodule our Umicom-GTK glue
git submodule add https://github.com/umicom-foundation/umicom-gtk external/umicom-gtk

# Commit submodule bindings
git add .gitmodules third_party external
git commit -m "chore(deps): add submodules (llama.cpp, tcc, glad, umicom-gtk)"
```

Checklist - Add `LICENSE` (MIT) at root; ensure each submodule license is preserved under their paths - Add `.gitattributes` for line endings; enable `*.c text eol=lf` (or consistent org standard)

---

3) Toolchain & Dependency Verifier (PowerShell)

> Run this first; it prints versions/paths and flags what's missing.

```
$tools = @('git','gh','cmake','ninja','meson','python','cl','clang','gcc','tcc','pkg-config')
foreach($t in $tools){
  $cmd = Get-Command $t -ErrorAction SilentlyContinue
  if($cmd){ Write-Host ("✔️ {0,-11} -> {1}" -f $t, $cmd.Source) -ForegroundColor Green }
  else    { Write-Host ("✖️ {0,-11} -> NOT FOUND" -f $t) -ForegroundColor Red }
}

# Print versions (no-fail)
cmake   --version 2>$null
ninja   --version 2>$null
meson   --version 2>$null
python  --version 2>$null
cl      2>$null
clang   --version 2>$null
gcc     --version 2>$null
tcc     -v 2>$null

# Expected directories we’ll create later
$expect = @('C:\gtk-build\gtk\x64\release\bin',
            'C:\gtk-build\gtk\x64\release\lib\pkgconfig',
            'C:\dev\curl\build\lib')
$expect | ForEach-Object {
  if(Test-Path $_){ Write-Host "✔️ exists: $_" -ForegroundColor Green }
  else            { Write-Host "⏳ will create: $_" -ForegroundColor Yellow }
}
```

> If `cl` (MSVC) is missing, install **Visual Studio Build Tools** (C++ workload) and run from **x64 Native Tools** prompt or import `vcvars64.bat`.

---

4) Build GTK4 & GtkSourceView from Source (Windows, Visual Studio)

> We use **gvsbuild** to build the GTK stack with MSVC (no MSYS2). Results land in `C:\gtk-build\gtk\x64\release`.

```
# 4.1 Get gvsbuild
Set-Location C:\dev
git clone https://github.com/wingtk/gvsbuild.git
Set-Location .\gvsbuild

# 4.2 Prepare Python venv
py -3 -m venv .venv
. .\.venv\Scripts\Activate.ps1
```

```
pip install -U pip
pip install -r .\requirements.txt  -q

# 4.3 Build GTK4 (and optionally GtkSourceView if available as recipe)
# Build can take a while; ensure disk space & VS Build Tools installed.
python .\build.py build gtk4
# If recipe exists for gtksourceview5 (varies by time), build it too:
# python .\build.py build gtksourceview5

# 4.4 Export environment for IDE builds (persist in script scripts/env-gtk.ps1)
$gtkRoot = "C:\gtk-build\gtk\x64\release"
[Environment]::SetEnvironmentVariable('PKG_CONFIG_PATH', "$gtkRoot\lib\pkgconfig", 'User')
[Environment]::SetEnvironmentVariable('Path',            "$env:Path;$gtkRoot\bin", 'User')
[Environment]::SetEnvironmentVariable('Lib',             "$gtkRoot\lib", 'User')
# Activate for current session too
$env:PKG_CONFIG_PATH = "$gtkRoot\lib\pkgconfig"
$env:Path           += ";$gtkRoot\bin"
$env:Lib             = "$gtkRoot\lib"

# 4.5 Sanity checks
pkg-config --modversion gtk4
# If gtksourceview5 was built:
# pkg-config --modversion gtksourceview-5
```

Notes - If GtkSourceView isn't packaged by gvsbuild at this moment, plan B is to build it with Meson against the GTK from `C:\gtk-build`. We can add a helper `scripts/build-gtksourceview.ps1` to automate fetch → meson → ninja → install into `C:\gtk-build`.
- Keep a cached `C:\gtk-build` between runs; our weekly artifact will bundle needed DLLs.

---

5) Build libcurl with native Windows TLS (Schannel)

> Objective: avoid OpenSSL; produce `libcurl.lib` + headers under `C:\dev\curl\build`.

```
Set-Location C:\dev
git clone https://github.com/curl/curl.git
Set-Location .\curl

# Configure (static lib; Schannel for HTTPS)
cmake -B build -G "Ninja" -DCMAKE_BUILD_TYPE=Release -DBUILD_SHARED_LIBS=OFF -DCURL_USE_SCHANNEL=ON
cmake --build build --config Release

# Optional: install to staged prefix (headers + lib)
```

```
New-Item -ItemType Directory -Force -Path C:\dev\curl\stage | Out-Null
cmake --install build --prefix C:\dev\curl\stage

# Update PATH/LIB hints for IDE builds
$env:CURL_ROOT = "C:\dev\curl\stage"
```

Linking tips - If linking **static** libcurl on Windows, add `-DCURL_STATICLIB` to compile defs and link `Ws2_32.lib`, `Crypt32.lib`, `Wldap32.lib` as needed.
- If you build a **DLL** instead, ensure the DLL sits next to `ustudio.exe` at runtime.

---

6) Configure & Build Umicom Studio IDE (CMake/Ninja)

```
Set-Location C:\dev\umicom-studio-ide

# Clean build directory
Remove-Item -Recurse -Force .\build -ErrorAction SilentlyContinue
New-Item -ItemType Directory -Force -Path .\build | Out-Null

# CMake configure (finds GTK via pkg-config; curl via config or find_package)
cmake -S . -B build -G "Ninja" -DCMAKE_BUILD_TYPE=RelWithDebInfo `
  -DCMAKE_PREFIX_PATH="${env:CURL_ROOT}" `
  -DPC_PATH="$env:PKG_CONFIG_PATH"

# Build
cmake --build build --config RelWithDebInfo

# Run (ensure GTK DLLs are on PATH)
Set-Location .\build
./ustudio.exe
```

CMake fixes (update your root `CMakeLists.txt`): - Add **GtkSourceView** pkg module (e.g., `pkg_check_modules(GSV REQUIRED gtksourceview-5)`) and link/include dirs accordingly. - Link **libcurl** to `ustudio` (remove stray `studio_fim_demo` unless you define that target).
- Generate `ustudio-resources.c` from `ustudio.gresource.xml` and `target_include_directories(ustudio PRIVATE ${CMAKE_BINARY_DIR})` for that generated file.

---

7) IDE Minimum Viable UI (GTK4 + GtkSourceView)

Window layout - Headerbar with **File / Edit / Build / AI / Help** menus - Main area: `GtkPaned` with **Project Tree** (left) and **Editor** (right — `GtkSourceView` in `GtkScrolledWindow`) - Bottom: **Output Console** (build/run output, LLM logs)

Editor bring-up checklist - Initialize `GtkSourceLanguageManager` and set language from filename - Enable line numbers, highlight current line, matching brackets - Wire **Open**, **Save**, **Save As** (use

`GFile*`, `GtkFileDialog`)
- Insert FIM result at cursor / saved mark; style AI insertions via `GtkTextTag`

---

8) LLM Backends

llama.cpp (embedded): - Build as part of solution (static lib) under `third_party/llama.cpp`
- Define `HAVE_LLAMA_H` and include `llama.h`; expose `ueng_llm_open/prompt/close` to use llama.cpp for local models (GGUF)
- Add IDE setting for model path + context size

OpenAI (HTTP): - Implement in `llm_openai.c` using libcurl (Chat Completions).
- Read API key from config or `OPENAI_API_KEY` env; model from app settings.

Ollama (local server): - Implement in `llm_ollama.c` via `POST /api/generate` (optionally `/api/chat`) to `http://127.0.0.1:11434`
- For **FIM**, send `prompt` and `suffix` (Codestral/CodeLlama style) and insert result into editor.

Config surface (GUI) - Provider: `Local (llama.cpp) | OpenAI | Ollama`
- Model: file path or model id
- Keys / host / port as needed

---

9) Compiler Switcher (C, C++, later Rust/Zig)

TinyCC (default quick-run) - Command: `tcc -run <file.c>`; capture stdout/stderr into Output Console - If TCC missing, show actionable message; allow user to set path in Settings

GCC/Clang (full build) - Single file compile: `gcc <file.c> -o <file.exe>` then run
- Project compilation (later): generate simple makefile or CMake preset

Rust/Zig (phase 2) - Rust: `cargo run` when `Cargo.toml` present
- Zig: `zig run main.zig`

---

10) PowerShell — Run/Build Helpers (drop into scripts/)

scripts\env-gtk.ps1

```powershell
param([string]$Root = 'C:\gtk-build\gtk\x64\release')
$env:PKG_CONFIG_PATH = "$Root\lib\pkgconfig"
$env:Path += ";$Root\bin"
$env:Lib  = "$Root\lib"
Write-Host "GTK env set for: $Root" -ForegroundColor Green
```

scripts\build-ide.ps1

```
. $PSScriptRoot\env-gtk.ps1
Set-Location (Join-Path $PSScriptRoot '..')
Remove-Item -Recurse -Force .\build -ErrorAction SilentlyContinue
cmake -S . -B build -G Ninja -DCMAKE_BUILD_TYPE=RelWithDebInfo
cmake --build build --config RelWithDebInfo
```

scripts\doctor.ps1

```
. $PSScriptRoot\env-gtk.ps1
# GTK present?
pkg-config --modversion gtk4
# Optional: run IDE and exit after smoke-init (future --smoke flag)
# .\build\ustudio.exe --smoke
```

---

11) Security & Secrets

- Never hardcode API keys. Provide `config.yaml` template committed as `config.example.yaml`.
- On first run, prompt to import keys → write to `%APPDATA%\Umicom\config.yaml`.
- Git ignore any real secrets; include `.env.example` for convenience.

---

12) Weekly Windows Release (ZIP)

Contents - `ustudio.exe` + required GTK DLLs
- `uaengine_cli.exe` (optional)
- `bin\tcc.exe` (if licensing approach OK), or instruct download on first run
- `third_party\*` license attributions
- `docs\QuickStart.md`

Automation - GitHub Actions (windows-latest) job: build GTK via cached gvsbuild or download prebuilt artifact; build IDE; collect DLLs; zip; upload release.

---

13) Milestones

1) **Week 1–2**: Fix CMake, build curl, build GTK via gvsbuild, app window opens, SourceView loads
2) **Week 3–4**: UAEngine library integrated in GUI; Open/Build/Serve content actions
3) **Week 4–6**: TCC “Run” and GCC/Clang “Build”; output console + error surfacing
4) **Week 6–8**: LLM: OpenAI + Ollama implemented; llama.cpp embedded; FIM insert flow
5) **Week 8–10**: Rust/Zig support, project templates, packaging & weekly ZIP
6) **Ongoing**: Docs, tests, CI, code style, contributor guide

---

14) Known Risks & Mitigations

- **GTK/GTKSOURCEVIEW builds are heavy** → use gvsbuild (MSVC) to standardize; cache artifacts between runs
- **DLL hell on Windows** → ship curated runtime set; prefer static curl; keep PATH minimal
- **TCC licensing (LGPL)** → prefer dynamic linking or provide source + build steps; document obligations
- **Large models (llama.cpp)** → keep model mgmt optional; integrate Ollama for an API path

---

15) Next Steps (Actionable)

1) Run **Section 2** (org, forks, submodules)
2) Execute **Section 3** verifier; install any missing tools
3) Build **GTK** via **Section 4**
4) Build **curl** via **Section 5**
5) Configure & build IDE (Section 6) → verify window opens
6) Implement LLM backends + editor features (Sections 7–8)
7) Wire compiler switcher (Section 9) and scripts (Section 10)
8) Prepare weekly Windows ZIP (Section 12)

---

16) Appendix — Example MIT Header Banner

```
/
*----------------------------------------------------------------------------
 * Umicom Studio IDE
 * File: ide/src/window.c
 * PURPOSE: Main application window (GTK4), editor + console scaffolding
 *
 * Created by: Umicom Foundation (https://umicom.foundation/)
 * Author: <Your Name>
 * Date: <YYYY-MM-DD>
 * License: MIT
 *---------------------------------------------------------------------------
*/
```

---

17) Appendix — Minimal SourceView Bring-up Snippet (C)

```
// create_source_editor(): returns a GtkWidget* (scrolled window containing
GtkSourceView)
GtkWidget* create_source_editor(void) {
    GtkSourceLanguageManager *lm = gtk_source_language_manager_get_default();
    GtkSourceBuffer *buf = GTK_SOURCE_BUFFER(gtk_source_buffer_new(NULL));
    gtk_text_buffer_set_enable_undo(GTK_TEXT_BUFFER(buf), TRUE);
    GtkWidget *view = gtk_source_view_new_with_buffer(buf);
    gtk_source_view_set_show_line_numbers(GTK_SOURCE_VIEW(view), TRUE);
    gtk_source_view_set_highlight_current_line(GTK_SOURCE_VIEW(view), TRUE);

    GtkWidget *scroll = gtk_scrolled_window_new();
    gtk_scrolled_window_set_child(GTK_SCROLLED_WINDOW(scroll), view);
    return scroll;
}
```

18) Appendix — PowerShell: Quick “Run C with TCC”

```
param([string]$File)
if(-not $File){ Write-Host "Usage: .\\run-tcc.ps1 .\\hello.c"; exit 1 }
$tcc = Get-Command tcc -ErrorAction SilentlyContinue
if(-not $tcc){ Write-Error "TinyCC not found. Install or add to PATH."; exit
2 }
& tcc -run $File 2>&1 | Tee-Object -Variable output | Write-Host
```

19) Prompts (LLM-Ready) — Examples

19.1 Implement OpenAI Backend (C, libcurl)

- Context: `uaengine/src/llm_openai.c` stub → implement `ueng_llm_open/prompt/close` using Chat Completions
- Constraints: single C file, libcurl only, read API key from env/config, robust error messages, MIT header, comments
- Deliverables: compilable C code + tiny usage doc in comments

19.2 Implement Ollama Backend (C, libcurl)

- Context: `uaengine/src/llm_ollama.c` stub → implement POST `/api/generate` with `prompt` and optional `suffix` for FIM
- Config: host/port/model in config; stream=false first; handle large outputs; MIT header

19.3 Embed llama.cpp (local)

- Context: `third_party/llama.cpp` static lib → `llm_llama.c` calls `llama_init_from_file`, prompt, read tokens; expose ctx size param
- Deliverables: CMake additions, error paths, test function `llm-selftest`

19.4 GtkSourceView FIM Insert

- Context: after AI returns code, insert at saved mark with highlight tag `ai-suggested` and scroll to reveal

(Full 1–3K word prompts per feature can be expanded on request — kept concise here to fit this Canvas.)

20) Resource Index (for engineers)

- GTK4 build & docs, GtkSourceView 5, gvsbuild, libcurl build guides, GitHub CLI usage, llama.cpp, Ollama API… (See chat message for full clickable references.)

End of Report — Part 1.

> If you want me to continue with deeper **feature prompts (1–3K words each)**, a **CI YAML**, and **ready-to-commit scripts & skeleton sources** in a downloadable ZIP, say **“continue – part 2”** and I’ll generate them next.