Umicom Foundation — Master Project Blueprint (Working Doc)

Owner: Umicom Foundation — Sammy Hegab
Scope: Current state, decisions, assets, priorities, next actions (Windows → Linux → RISC-V).
Note: This is a *living* master reference compiled from your uploaded files and our agreements in this session. No external web research is included here.

---

1) Executive Summary

We are building a reproducible, open-source IDE and tooling stack centered on **C** and **Assembly**, with clean interop for **C++**, **Rust**, and **Zig**. The flagship desktop application is **Umicom Studio IDE (GTK4)**, integrating **UAEngine** workflows and **LLM** backends (local **llama.cpp**, plus API adapters). We will: - **Standardise** the repository structure and formatting across all projects. - **Avoid MSYS2/vcpkg** for final deliverables; prefer **fork/submodule + build-from-source** under `C:\dev`. - Use **GCC (MinGW-w64)** as the **default** compiler on Windows, **CMake** as **primary** build system. - Create **Umicom-GTK** (GTK stack meta-repo) and **Umicom-LLVM** (LLVM/Clang/LLD fork), both from source. - Ship a **Windows PowerShell** bring-up first; then replicate on **Linux** and plan for **RISC-V**.

Key blockers today: the IDE doesn't compile on Windows; GTK headers/libs are missing; CI fails. This blueprint documents what we have, what we've decided, and a concrete plan to reach a compiling IDE shell quickly and cleanly.

---

2) Umicom Foundation — Introduction

Vision: An open, ethical, high-performance engineering stack—IDEs, compilers, runtimes, and tools—that empowers authors, engineers, and humanitarian projects.
Mission: Deliver a modern, open-source desktop IDE (Umicom Studio IDE) that integrates UAEngine, local/API LLMs, and a bundled compiler toolchain—*all reproducible from source* and portable to Windows, Linux, and RISC-V.

Core tenets - Open-source only in core deliverables.
- From-source policy: fork/submodule each dependency; pin revisions; script builds.
- Determinism: documented steps must reproduce identical outputs on clean machines.
- Secrets via environment (with `.env.template`); never commit credentials.
- Hygiene: mandatory MIT header on every file; `clang-format` enforced.

---

3) Project Portfolio — Overview & Intent

- **Umicom Studio IDE (UStudio):** GTK4 desktop IDE that integrates UAEngine and LLM backends; compiler runner for C/ASM (+ C++/Rust/Zig as needed).
- **UAEngine (Umicom AuthorEngine AI):** C-based CLI/library for ingest→build→export workflows; to be embedded/invoked within the IDE.

- **UMICC:** Umicom compiler initiative (C/C++/ASM to start); integrated later as it matures.
- **UAI language & tools:** language + extensions (e.g., VS Code/CLI).
- **Umicom-LLVM:** fork of LLVM/Clang/LLD built from source (Windows first; Linux & RISC-V targets next).
- **Umicom-GTK:** meta-repo for GTK stack (glib/gtk/pango/cairo/harfbuzz/gdk-pixbuf/freetype/libpng/fribidi), all from source.
- **Ecosystem apps (catalogue):** Bank, Exchange, FundMe, Social Engine, OS, Framework, Office Apps, Medical PRS, Weight-Loss Tracker, Website, etc.

---

4) Decisions & Standards (Locked-in)

- **Languages:** C & Assembly first. Allow C++, Rust, Zig where they interop cleanly with C.
- **Compiler (Windows host): GCC (MinGW-w64)** as **default**.
- **Build system: CMake** primary (Meson secondary).
- **GUI toolkit: GTK4** with **GtkSourceView** in the editor.
- **Local LLM target: llama.cpp** first; compare **llmcpp** for C-friendly integration.
- **Bundled compilers:** Include **TinyCC** as a submodule; **disabled by default**. Add **UMICC** later.
- **From-source policy:** Prefer **fork/submodule + build-from-source** in `C:\dev` for GTK, LLVM, etc.
- **CI:** GitHub Actions **windows-latest** primary; **ubuntu-latest** secondary.
- **Secrets:** environment variables; ship a `.env.template`.
- **Code style:** repository-root `.clang-format` + **MIT header block** at top of every source/header/script/doc.
- **Canonical layout (reference):**

```
C:\dev\<project>
  build\            # out-of-source builds
  cmake\            # toolchain cmake modules
  dist\             # ship/pack artifacts (bin, lib, include, tools,
pkgs)
    bin\ lib\ include\ tools\ pkgs
  docs
  include
  scripts\          # PowerShell/Bash
  src
  third_party\      # submodules (pinned)
  ui\               # .ui/.css/.gresource.xml
  .clang-format
  CMakeLists.txt
  README.md
```

---

5) Master Inventory — Files & Assets (This Workspace)

Below is the consolidated list of files and assets you uploaded or referenced that we currently have recorded. Items marked with `*` exist but could not be opened here (e.g., certain .ui/.zip/.yaml/.meson variants):

5.1 Umicom Studio IDE — Source & Build

- **Sources/Headers:** `app.c`, `app.h`, `editor.c`, `editor.h`, `window.c`, `window.h`, `settings.c`, `settings.h`, `tasks.c`, `tasks.h`, `logging.c`, `logging.h`, `gtk_smoke.c`, `window_chat_integration.c`, `studio_codestral_fim.c`, `main.c` (multiple variants recorded), `ustudio.gresource.xml`, `main.ui*`, `settings.ui*`.
- **Build/Meta/Scripts:** `CMakeLists.txt`, `meson.build*`, `meson.build.bak*`, `README.md`, `README-Codestral.md`, `index.html`, `openapi.yaml*`, `build-umicc.ps1*`, `build-umicc.sh`, `init-submodules.ps1*`, `init-submodules.sh`.
- **Branding/UI:** `com.umicom.ustudio.svg`, `logo.svg`, `logo2.svg`, `thumbnail_UMICOM LOGO .jpg`.
- **Screenshots:** VS Code GTK include error; local drive layout.

5.2 UAEngine — Source & Build

- **Sources/Headers:** `fs.c`, `fs.h`, `serve.c`, `serve.h`, `common.c`, `common.h`, `llm.h`, `llm_openai.c`, `llm_llama.c`, `llm_ollama.c`, `main.c`, `ueng_config.c`, `config.h`, `version.h`.

5.3 Documentation — Strategy & Chapters

- **Roadmaps/Papers:**
- *Umicom Studio & Author Engine – Comprehensive Guide and Roadmap.pdf*
- *Umicom Studio IDE – Initial Roadmap and Setup.pdf*
- ***Umicom Foundation Project Status & Integration Roadmap**.pdf*
- **Chapters:**
- *Chapter_01_Introduction_and_Development_Setup.docx*; *Chapter_01_Full_Introduction_and_Development_Setup.docx*; *Chapter_01_… .pdf*
- *Chapter_02_Full_Engineering_and_Workflows.docx*
- *Chapter_03_UStudio_GTK4_Architecture_and_Integration.md*
- *Chapter_06_Ingest_OCR_Normalisation_and_Conversion_Pipeline.md*
- *Chapter_12_Authoring_UX_and_Accessibility.md*
- *Chapter_14_AI_Prompting_Guardrails_and_Evaluation.md*
- *Chapter_17_Distributed_Builds_and_Remote_Execution.md*
- *Chapter_19_Security_Secrets_and_Key_Management.md*
- *Chapter_23_Observability_and_Telemetry.md*
- *Chapter_24_Data_Pipelines_and_ETL.md*
- *Chapter_25_Security_Hardening_and_Threat_Modeling.md*
- *Chapter_31_Data_Contracts_and_Schemas.md*
- *Where we are.docx*

5.4 Archives & Bundles

- *umicom-studio-ide_gtk4_app_v7.zip* (not readable here, but recorded).

---

6) External Repositories & Links (Tracked)

Tracked for inspiration/integration. We will fork and submodule those we adopt under the Umicom organization. - **Primary:** https://github.com/umicom-foundation/umicom-studio-ide (plus other Umicom org repos you listed).

- `umicom-foundation/umicom-authorengine-ai`
- **Tooling & Langs:** `nature-lang/nature`, `pygame/pygame`, `Dav1dde/glad`, `c3lang/c3c`, `clibs/clib`, `carbon-language/carbon-lang`, `vlang/v`.
- **LLM & Runtime:** `lucaromagnoli/llmcpp`, `ollama/ollama`, `ggml-org/*`, `lmstudio-ai/*` (and lmstudio.ai).
- **Profiles/Sites:** `github.com/sammyhegab`, `github.com/xcreatelabs/*`, `playir.com`.
- **Search streams:** GUI+C (language:C), C language (language:C), Bitcoin (C++ repos by forks).

---

7) Umicom Studio IDE — Current State & Diagnosis

Symptoms
- Compilation fails on Windows with unresolved `gtk/gtk.h` (and other GTK symbols).
- GitHub CI also fails; dependency discovery not deterministic.
- Duplicated or stub files (`main.c` variants) risk build confusion.
- Resource pipeline (GResource) not guaranteed to run; UI XMLs may not be embedded/loaded.

Likely root causes
- GTK stack and/or include/library paths not installed and pinned for Windows.
- Non-canonical build setup (CMake vs Meson) causes ambiguity.
- Missing/disabled GtkSourceView integration.
- Absent coding standards (headers/format) lead to drift and confusion.

Constraints (as agreed)
- **PowerShell-only** instructions on Windows.
- Avoid MSYS2/vcpkg in final deliverables; rely on **from-source** forks.
- GCC default; CMake primary; GTK4+GtkSourceView; TinyCC present but off; LLM via llama.cpp (local) and API adapters.

---

8) Normalised Repository Structure (All Projects)

```
C:\dev\<project>
  build
  cmake
  dist
    bin\ lib\ include\ tools\ pkgs
  docs
  include
  scripts
  src
  third_party
  ui
  .clang-format
  CMakeLists.txt
  README.md
```

Notes: - `third_party/` hosts **forked submodules** (pinned to commits) for GTK stack, GtkSourceView, llama.cpp/llmcpp, UAEngine, TinyCC, etc.
- `dist/` is the **consumption prefix** for built artifacts used by the IDE (headers/libs/tools).
- `scripts/` contains **PowerShell** helpers (`bootstrap.ps1`, `verify-gtk.ps1`, `verify-toolchain.ps1`, etc.).

---

9) Standards — Headers, Formatting, Security

MIT Header (apply to every source/script/doc)

```
/
*------------------------------------------------------------------------------
 * <Project Name>
 * File: <relative/path/filename>
 * PURPOSE: <one-line purpose>
 *
 * Created by: Umicom Foundation (https://umicom.foundation/)
 * Author: Sammy Hegab
 * Date: <YYYY-MM-DD>
 * License: MIT

*------------------------------------------------------------------------------
 * QUICK START / NOTES:
 *   - Follows Umicom's coding standard and clang-format style.
 *   - Secrets are loaded from environment variables; never commit keys.
 *   - See docs/ for architecture diagrams and additional guidance.
 *-----------------------------------------------------------------------------
*/
```

Formatting: enforce `clang-format` from a root `.clang-format`.
Secrets: `.env.template` lists variables; do not commit real keys; redact in logs.
Licences: keep upstream `LICENSE` files under `third_party/<lib>/LICENSE`; aggregate notices in `docs/`.

---

10) Bring-Up Strategy (Windows → Linux → RISC-V)

Phase A — Windows IDE shell up
1. Canonicalise **CMake** (Windows).
2. Implement **GtkSourceView** editor (open/save; tabs; status bar).
3. Wire **GResource** to embed UI XML/assets.
4. Add **compiler tasks** (GCC default; TinyCC toggle off by default).
5. Provide **UAEngine** action(s) via menu + task runner.
6. Expose **LLM** panel with local runner stub + API adapter stub.

Phase B — From-source stacks
- **Umicom-GTK** meta-repo: submodule forks of glib/gtk/gdk-pixbuf/pango/cairo/harfbuzz/freetype/

libpng/fribidi → build to `C:\dev\umicom-gtk\dist`. - **Umicom-LLVM**: fork llvm/clang/lld → build to `C:\dev\umicom-toolchains\llvm` (Clang/LLD available; GCC remains default for now).

Phase C — Cross-platform
- Replicate on Linux (same structure).
- Define RISC-V cross targets; add CI jobs; minimal IDE build against riscv64 toolchains.

---

11) Detailed Milestones & Acceptance

M0 — Baseline (Windows): - CMake canonical; one `main()`; compiles to a window; GResource embedded; GtkSourceView working; run **Hello World** via GCC task; UAEngine subprocess call prints to console pane.
- *Done when:* IDE launches and performs the above on a clean Windows machine with provided scripts.

M1 — From-source stacks: - Umicom-GTK & Umicom-LLVM meta-repos live with PowerShell build scripts; first Windows release ZIPs.
- *Done when:* IDE consumes our `dist/` artifacts with no external package managers.

M2 — IDE extensions: - Compiler picker (GCC/TinyCC); richer tasks; LLM adapters (local/API); logs panel; improved file tree.
- *Done when:* All features are usable end-to-end with sample projects.

M3 — Cross-platform: - Linux bring-up; RISC-V cross builds; CI green on both OSes; publish cross artifacts.
- *Done when:* We can produce and run IDE builds on both platforms; RISC-V cross artifacts available.

---

12) Plan of Action — Near-Term Task List

1. **Repo cleanup:** remove duplicate/stub files; unify `main.c`; add `.clang-format`; apply MIT headers.
2. **CMake first:** ensure a single, working `CMakeLists.txt`; add `CMakePresets.json`.
3. **UI resources:** finish `ustudio.gresource.xml`; ensure `main.ui`/`settings.ui` are compiled and loaded.
4. **Editor:** integrate GtkSourceView; implement basic file operations; status bar; tabs.
5. **Tasks:** add PowerShell scripts for compile/run; bind to menu/toolbar + key shortcuts.
6. **UAEngine:** add menu/command to call UAEngine build/export; capture output.
7. **LLM:** define `llm_adapter.h`; wire local/HTTP stubs; read keys from env.
8. **CI:** add `windows-latest` workflow that mirrors our PowerShell steps; add `ubuntu-latest` job.
9. **Docs:** keep this blueprint in `docs/` and update as we progress.

---

13) What We Will Research (once you say “start”)

- **GTK stack pinning on Windows:** exact commits, build flags, and `dist/` layout for Umicom-GTK.
- **GtkSourceView versions:** best match for our GTK pin; Windows build peculiarities.

- **llama.cpp vs llmcpp:** API surface, threading, memory footprint; easiest path to embed into a C IDE.
- **Clang/LLD vs GCC defaults:** impact on diagnostics, speed, and Windows packaging.
- **RISC-V targets:** recommended triplets and minimal runtime deps for IDE shell.
- **Security posture:** secrets flow, token scoping, rate limits; guardrails for prompting (see chapters 14 & 19).
- **Telemetry:** minimal schema for logs/metrics/traces; opt-in policy (see chapter 23).
- **Data contracts:** serialisation formats for settings, tasks, LLM prompts (see chapter 31).
- **Distributed builds:** staged caching and remote execution options (chapter 17).

---

14) Open Questions (to lock before deep work)

1. **Umicom-GTK governance:** meta-repo with submodules (patches in branches) vs single overlay with patch series?
2. **Umicom-LLVM targets:** confirm initial set — `x86_64-pc-windows-gnu`, `x86_64-linux-gnu`, `riscv64` (ELF vs Linux).
3. **Artifacts cadence:** publish weekly source-built ZIPs for GTK/LLVM (Windows first)?
4. **IDE layout v0:** single-window with tabs (Editor | Console/Build | Chat/LLM) OK for first release?
5. **Compiler picker default:** GCC is default; TinyCC present but OFF — confirm UX toggle location (status bar + settings?).
6. **Bundle strategy:** vendor source as submodules only (no binaries) and publish releases to be consumed by IDE scripts?

---

15) Risks & Mitigations

- **Heavy builds (GTK/LLVM):** mitigate with caching and split components; attach artifacts to GitHub releases.
- **Windows path/toolchain complexity:** use absolute paths and CMake toolchain files; generate verbose link logs.
- **Licensing diligence:** store upstream licences intact; track in `docs/THIRD_PARTY_LICENSES.md`.
- **Secrets & privacy:** env-vars only; never log keys; redact prompts containing PII; add guardrails (chapter 14).

---

16) Appendix — Quick Reference Snippets

PowerShell (scaffold):

```
$DevRoot = 'C:\dev'
$Studio  = Join-Path $DevRoot 'umicom-studio-ide'
$GTK     = Join-Path $DevRoot 'umicom-gtk'
$LLVM    = Join-Path $DevRoot 'umicom-llvm'
$Tools   = Join-Path $DevRoot 'umicom-toolchains'
New-Item -ItemType Directory -Force -Path $DevRoot,$Studio,$GTK,$LLVM,$Tools
| Out-Null
```

```
# Clone (replace with Umicom forks)
git clone https://github.com/umicom-foundation/umicom-studio-ide.git $Studio

# CMake build (placeholder until Umicom-GTK/Umicom-LLVM artifacts are available)
New-Item -ItemType Directory -Force -Path (Join-Path $Studio 'build') | Out-Null
cmake -S $Studio -B (Join-Path $Studio 'build') -G "Ninja" -DCMAKE_BUILD_TYPE=Debug -DCMAKE_EXPORT_COMPILE_COMMANDS=ON
cmake --build (Join-Path $Studio 'build') -v
```

.clang-format (starter):

```
BasedOnStyle: LLVM
IndentWidth: 2
TabWidth: 2
UseTab: Never
ColumnLimit: 100
BreakBeforeBraces: Allman
AllowShortIfStatementsOnASingleLine: false
SortIncludes: true
```

LLM adapter surface (sketch):

```
// include/llm_adapter.h
#pragma once

typedef struct {
  int (*init)(void);
  int (*set_model)(const char* path);
  int (*prompt)(const char* text, void (*on_token)(const char* piece));
  void (*stop)(void);
  void (*shutdown)(void);
} llm_adapter_t;
```

17) One-line Summary

We now have a single, agreed plan to get **Umicom Studio IDE** compiling on Windows with **CMake**, embed **UI resources**, bring in **GtkSourceView**, wire **compiler tasks** and **UAEngine**, and prepare **from-source** stacks (**Umicom-GTK**, **Umicom-LLVM**) for long-term control. When you say **start**, we'll proceed with the deep, project-specific research and produce the granular PowerShell guide and working build artifacts.