**Optimizor 1.0 — Build Plan & Order of Operations (UE5.6) 13/08/25**

A pragmatic, macro→micro plan to deliver a stable, generalist optimization plugin (C++ + Python) with fast iteration and strong safety.

**0) Guiding Principles**

* **Macro→Micro**: lock the user-facing flow and config schema first; implement subsystems behind it.
* **Single Canonical Config**: JSON schema shared across UI ⇄ C++ ⇄ Python. Version it.
* **Safe By Default**: dry-run, change caps, revert snapshots, collections for review.
* **Incremental & Testable**: each milestone shippable, with automated sanity checks.
* **No dead UI**: any surfaced control must be wired into a no-op-safe path on day 1.

**1) Foundations (Week 1)**

**Goal:** Solid plugin skeleton, one tab, presets plumbed end-to-end, Texture pipeline callable.

**1.1 Module & Tab**

* Create Optimizor editor module (keep TextureDoctor code but mount under Optimizor).
* Nomad tab: SOptimizorTab with panes: Textures | Meshes | Materials | Reports.
* Hook LevelEditor Tools menu ➜ opens tab.

**1.2 Canonical Config Schema v1**

* Freeze initial JSON keys (dry\_run, max\_changes, include/exclude, target, preset, texture/mesh/material subtrees).
* Add schema\_version (int) and run\_id (timestamp string) added by UI.
* C++: FOptimizorConfig (UStruct) ⇄ JSON via FJsonObjectConverter.
* Python entry points accept same JSON per tool: textures.run(cfg), meshes.run(cfg), materials.run(cfg).

**1.3 Python Bridge**

* FOptimizorPythonBridge::Run(tool, JsonPath, bAutoFix) ➜ loads module, importlib.reload, executes.
* Standard stdout markers: [[REPORT\_JSON]] { ... } for compact summaries.
* Error channeling: capture exceptions and surface to UI toast + Report pane.

**1.4 Reports & History**

* Saved/Optimizor/History/<run\_id>/{report.txt, report.json, changes.csv, revert.json}.
* Collections: Optimizor\_Changed\_<run\_id> and Optimizor\_Skipped\_<run\_id>.
* Report pane reads history list and opens folders.

**Definition of Done (DoD):**

* Open tab ➜ select preset ➜ Scan Textures ➜ see modal summary + history entry.
* Auto-Fix writes revert snapshot and respects Max Changes.

**2) Texture System Hardening (Week 2)**

**Goal:** Ship-quality Texture optimizer (existing TextureDoctor refined to schema + UI).

**2.1 Consolidation & Fixes**

* Remove duplicate gather\_textures(); keep fast list-assets path with AR fallback.
* Align ACTIVE\_PRESET\_LABEL naming UI⇄py; ensure label appears in reports.
* Fix file save helper names in Slate panel and BOM-less UTF-8 writes.
* Ensure ExecPython fallback works when module handle isn’t present.

**2.2 Policy & Gates**

* Enforce APPLY\_ONLY + Conservative mode.
* Exceptions: path/name tags, skip collection.
* Inference from materials (cap referencers N; make optional per preset).
* Memory impact estimator (before/after) in report.

**2.3 UX Polish**

* “Explain change” tooltip per row (hook reasons map).
* Preset tooltips: best-practice guidance (non-engine changes).

**DoD:**

* Runs on sample projects (PC/Console/Mobile/VR/UI-heavy) with <1% error rate, responsive SlowTask.

**3) Mesh System (Weeks 3–4)**

**Goal:** LOD/Nanite/lightmap/collision passes with dry-run and reversible changes.

**3.1 C++ Subsystem**

* FMeshDoctor (C++) using IMeshReduction/NaniteSettings.
* Read LODGroup, apply policy order: **Group → Caps → Screens**.
* Auto-count LODs from triangle counts + screensize curve.
* Nanite on/off per preset; fallback screen size control.

**3.2 Lightmap & Collision**

* Generate Lightmap UVs within min/max res; detect overlaps, warn if baking risk.
* Collision: generate simple convex if none (skip if tagged).

**3.3 Reporting**

* Per-mesh: base triangles, per-LOD triangles %, Nanite enabled, lightmap res, collision type.

**DoD:**

* Dry-run surfaces accurate deltas; Auto-Fix creates revert snapshot for mesh settings.

**4) Material System (Week 5)**

**Goal:** Static-cleanup & diagnostics without altering authored look unless requested.

* Scan graphs for DX/GL normal mixing; list offenders.
* Detect redundant samplers; suggest consolidation.
* Find dead static switches; optional auto-prune.
* Packed map hints (\_orm, \_rma): verify channel swizzles.

**DoD:**

* CSV of material issues with safe auto-fix options gated by APPLY\_ONLY.

**5) Presets & Best-Practice Notes (Week 6)**

**Goal:** Generalist coverage + embedded guidance.

**5.1 Ship Presets**

* **PC High, Console Balanced, Mobile Low, Switch, VR, AR, Cinematic, UI Crisp, Archviz, Stylized Mobile, Low-Mem QA**.
* Each preset defines: texture caps, LOD strategy, Nanite policy, lightmap budget, apply-only gates, change cap.

**5.2 Preset Notes (non-engine guidance)**

* Color space, authoring DPI, atlas/packing, trim sheets, shader feature toggles, post-process hints.
* Shown inline in the UI and exported to report header.

**DoD:**

* Switching presets updates UI, JSON, and report label consistently.

**6) QA, CI, and Samples (Week 7)**

**Goal:** Prevent regressions and validate on real content.

* Commandlet/Headless: OptimizorCMD -Config=<json> -Target=Textures|Meshes|Materials.
* Test maps (open world, VR template, mobile-forward, UI heavy).
* CI checks: fail build if asset budgets exceed preset.
* Performance: cache scans; incremental re-run with include paths.

**7) UX/DevEx Polish (Week 8)**

**Goal:** Delight and productivity.

* History pane: rerun with last config; “Revert last run”.
* Perforce changelist helper (optional).
* Hot-list: most-referenced textures/materials.
* Tooltips for every control; hover docs.

**8) Engineering Practices & Repo Structure**

/Plugins/Optimizor/

Source/

OptimizorEditor/ # module, tab, UI

OptimizorCore/ # shared structs, config, report svc

TextureDoctor/ # texture C++ shims (if any)

MeshDoctor/

MaterialDoctor/

Python/

textures/texture\_doctor.py

textures/texture\_doctor\_entry.py

meshes/mesh\_doctor.py

materials/material\_doctor.py

Config/Presets/\*.json

Resources/

* **Branching:** main (stable), dev/\* per subsystem, PRs with report artifacts attached.
* **Coding Standards:** no dead controls, every write wrapped with revert capture, verify-after-write optional, unit tests for JSON IO.

**9) Risk Register & Mitigations**

* **Python enum drift**: tolerant resolver (already present) + unit tests.
* **Asset saves blocked by source control**: Auto-checkout flag + error surfaced per asset.
* **Long scans**: path filters + incremental cache + HEARTBEAT progress.
* **Preset creep**: schema\_version; keep backward-compat in entry points.
* **Material rewrite risk**: default to scan-only; opt-in auto-fix.

**10) Immediate Tasks (Next 2–3 Coding Sessions)**

1. Create OptimizorEditor module + Nomad tab scaffolding (reuse current module patterns).
2. Implement FOptimizorConfig + JSON read/write + schema\_version.
3. Build Python bridge wrapper + compact report parsing.
4. Port TextureDoctor UI into Textures pane; wire to canonical JSON; fix duplicates & save helper.
5. Report pane listing history, open-folder, create collections.

**Exit Criteria:** Scan/Auto-Fix for textures works through Optimizor tab and writes history.