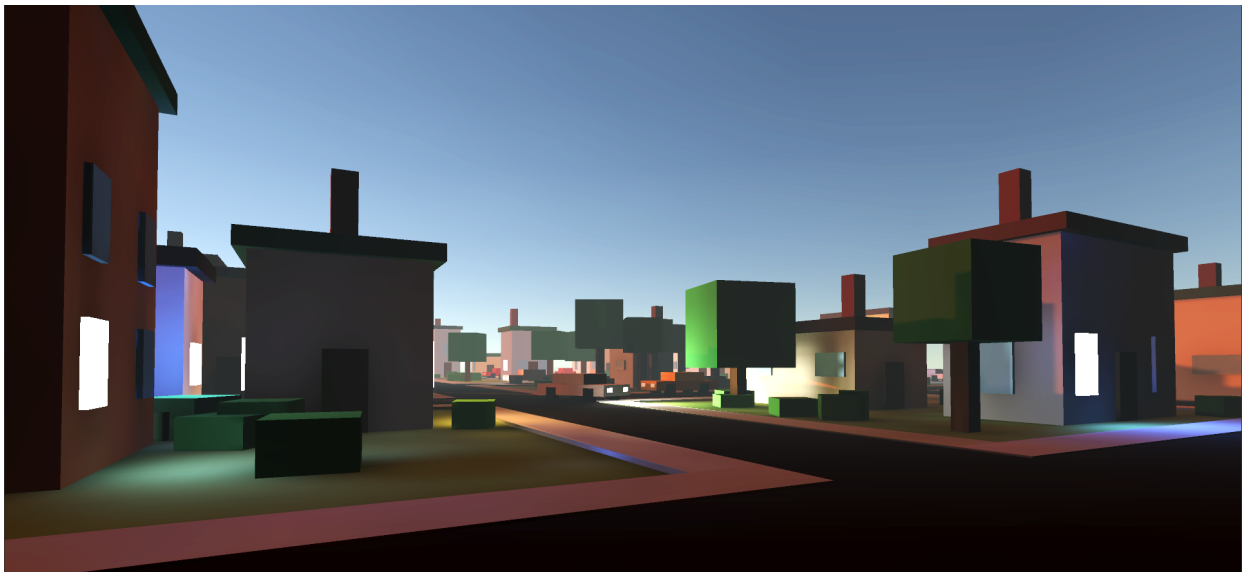
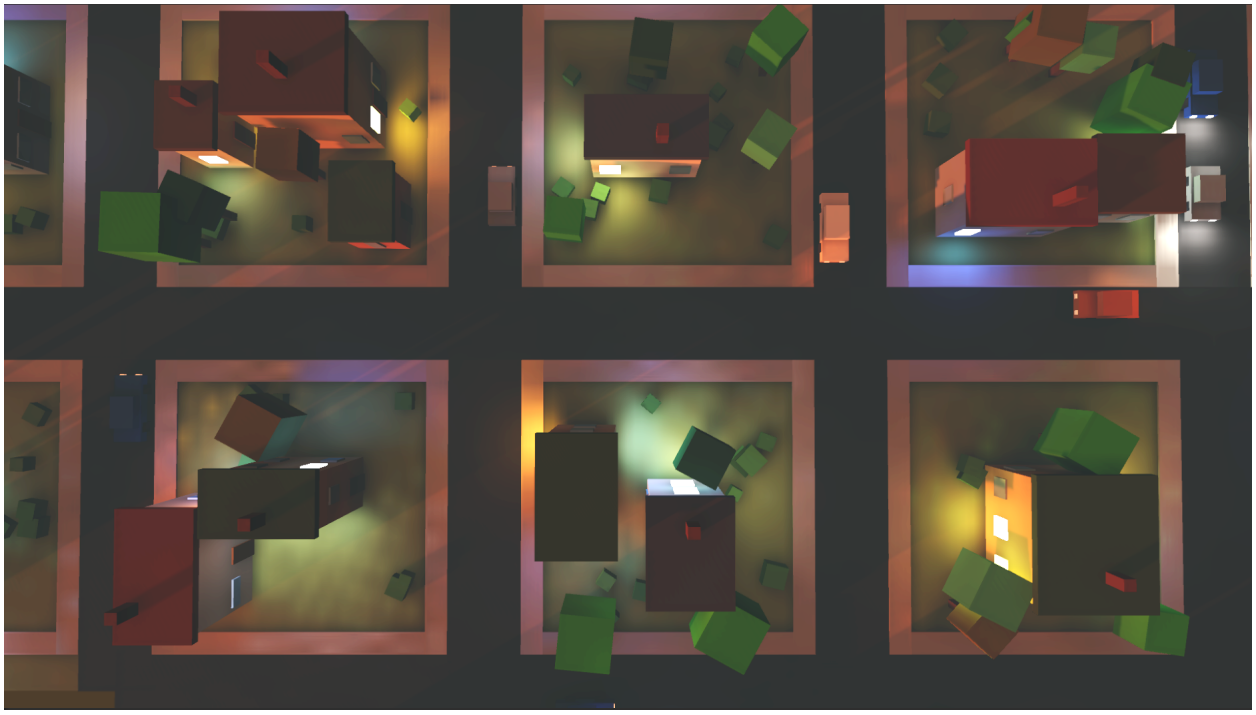
Runtime Lightmap Baker

Quick guide & API reference





Introduction

Runtime Lightmap Baker bakes your scene's lighting — soft shadows, bounced light, and glow — directly inside the running game, on the GPU.

Why it matters to your game: you get the rich, grounded look of baked global illumination WITHOUT Unity's editor bake step, and it works where the built-in lightmapper can't — including the browser (WebGL) and mobile. No pre-baked lighting data is shipped in your build; it's computed live.

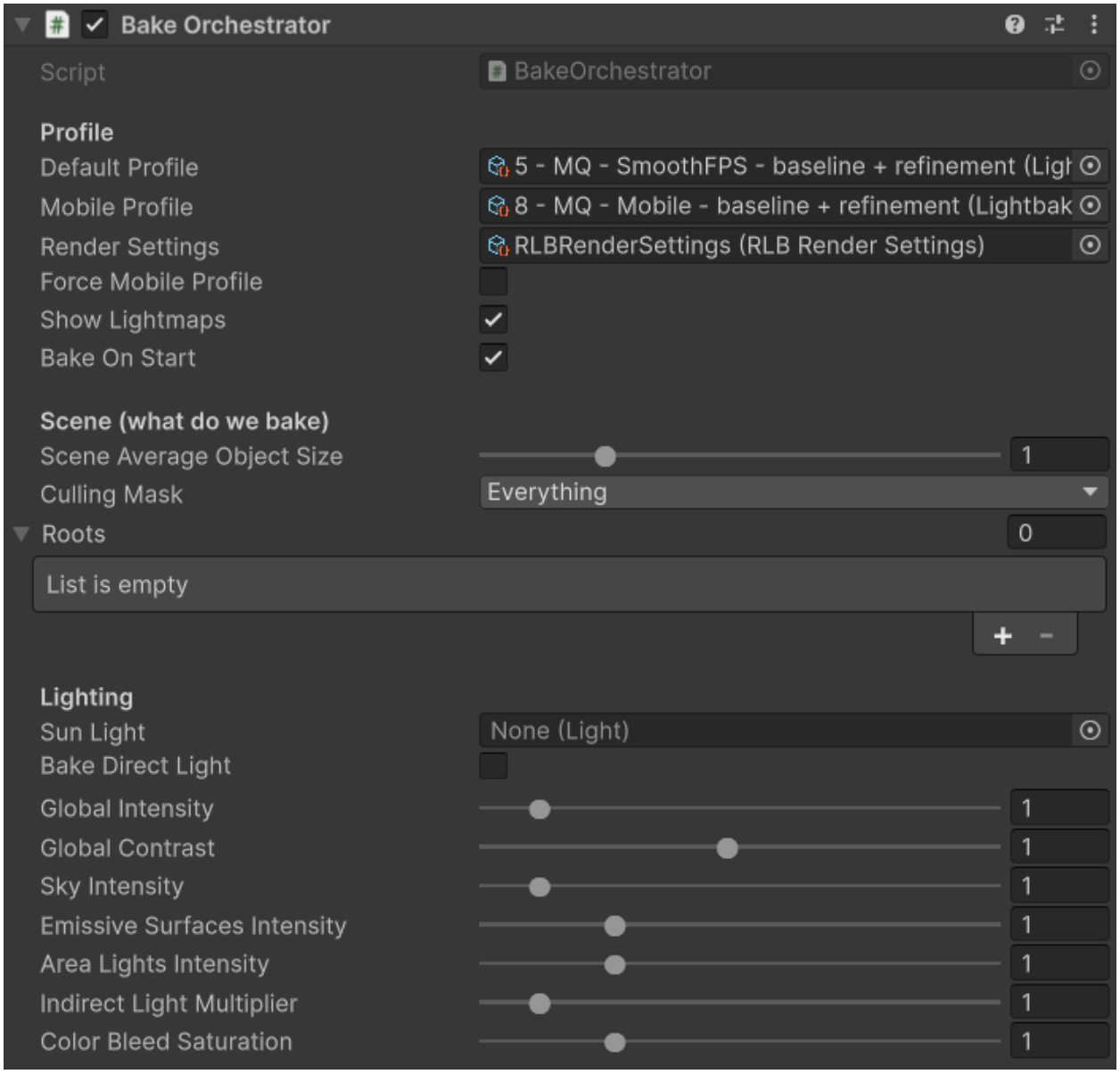
This guide is written for artists and designers. You do not need to understand the tech — every setting below leads with "why it matters to your game" and a plain rule of thumb.

1. QUICK START — LIGHT YOUR SCENE IN 3 STEPS

STEP 1 — Add the brain.

Add Component > Runtime Lightbaker > Bake Orchestrator.

This one component runs the whole bake.



STEP 2 — Pick a Quality Profile.

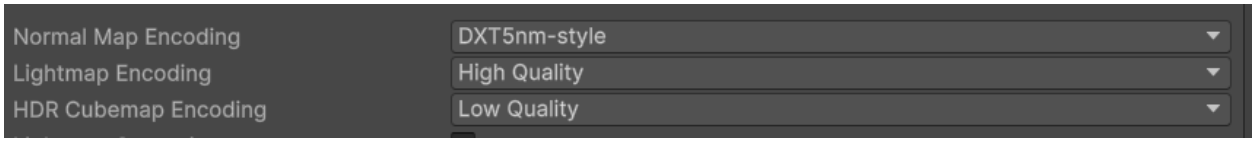
The Profile slot is the heart of the tool. Drag in one of the ready-made profiles (see section 2 — **“5 - MQ - SmoothFPS - baseline + refinement”** one is a safe start).

STEP 3 — Choose what to bake.

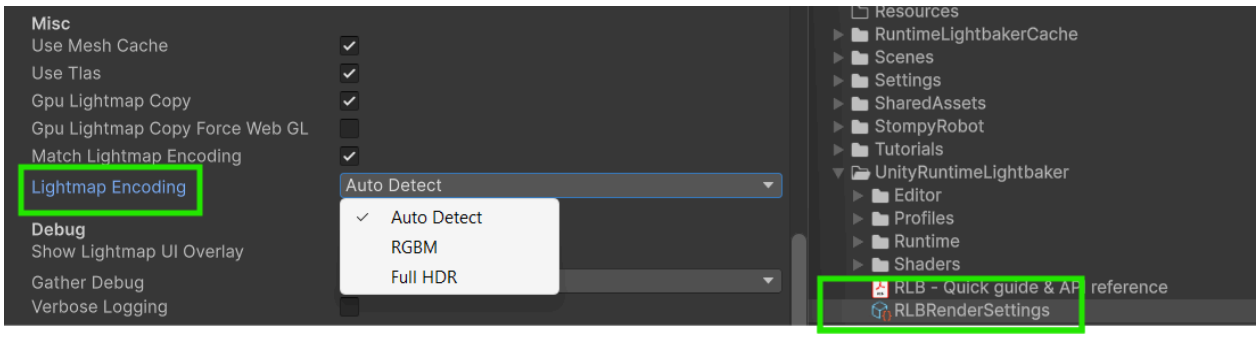
Leave "Roots" empty to bake the whole scene (the usual case). Or drag in specific parent objects — e.g. your level chunks — to bake only those. **Less is more!** The more you add, the more performance and VRam it will consume!

STEP 4 — Check your Lightmap Encoding (Mobile only)

On **Android** and **iOS** it is recommended to leave the quality at “High Quality” because internally the lightmaps are rendered as HDR anyways and this prevents darkening of the lightmaps.



If your project can’t have it set to “High Quality” and **your lightmaps turn dark**, consider playing around with the **“RLBRenderSettings -> Lightmap Encoding”** setting.



STEP 5 — Press Play.

It bakes automatically on start. Watch the scene light up. Toggle **"Show Lightmaps"** on the component at any time to compare before/after instantly, with no re-bake.

2. BAKE PROFILES

This tool is PROFILE-DRIVEN. A Bake Profile bundles every quality and performance setting into one asset, so almost all of the time you just pick a profile and you're done.

TWO PHASED BAKE:

• **BASELINE** — a fast first pass that lights the whole scene in seconds. Usable straight away, but a little grainy and edges aren't cleaned up yet.

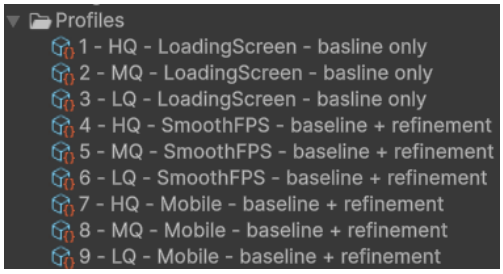
• **REFINEMENT** — extra passes that run quietly in the BACKGROUND afterward, removing grain and tidying seams until it looks final. The game stays fully playable the entire time.

USE-CASE

• **LoadingScreen** — a big first pass and NO background polish. Bakes a full-quality result in one fast burst. Meant to run HIDDEN BEHIND A LOADING SCREEN, right after your level has finished spawning: the player never sees the bake, so you spend the whole budget up front.

• **SmoothFPS** — a quick first pass THEN a background polish. For interactive/live setups where the environment changes and you want to re-bake WHILE the game keeps running at a smooth frame rate: lighting appears immediately, then quietly sharpens without hitching gameplay.

• **Mobile** — same quick-pass-then-polish approach, tuned for phones, tablets, and other low-power hardware.



HOW TO CHOOSE

1. Baking behind a loading screen after a level spawns? → a **LoadingScreen** profile.

2. Re-baking live while the game is playing (changing scenes)? → a **SmoothFPS** profile.

3. Shipping to phones / VR / low-power devices? → a **Mobile** profile. Then pick the Quality tier (HQ / MQ / LQ) your hardware can afford.

3. SPECIAL LIGHTS & GLOWING SURFACES

AREA LIGHTS (soft light from panels & windows)

Why it matters to your game: soft, realistic light from a panel, strip, or window that plain point/spot lights can't fake.

How: Unity's built-in area lights can't be baked by a runtime baker directly (and would leak a stray glow in a build), so they're replaced by a bakeable stand-in component ("RLB Area Light"). Two ways to get one:

• Convert existing lights (quick): run

Tools > Runtime Lightbaker > Convert Area Lights To Surrogates ONCE per scene. It swaps each Unity area light for a stand-in with the same size, color, and intensity. Re-run it whenever you add new area lights.

• Add it yourself (full control): put the light where you want it, then

Add Component > Runtime Lightbaker > RLB Area Light, and set the shape, size, color, and intensity by hand. Best when you want to author the baked light exactly, independent of any Unity light.

EMISSIVE SURFACES

How: select the glowing object, then **Add Component > Runtime Lightbaker > RLB Emissive Surface.**

It automatically picks up the material's emission color; tune Emission Intensity to taste. A wireframe box in the Scene view shows the light it casts. Use the global Emissive Surfaces Intensity on the Bake Orchestrator to dial ALL of them at once. (Leave Bounds Type on "Mesh Renderer" — it's the predictable, recommended choice.)

4. PROJECT SETUP — BEFORE YOU BUILD

Why it matters to your game: baking works in the editor out of the box, but a BUILD needs a couple of project settings in place — otherwise the lighting can come out blank or wrong in the build while looking fine in the editor. One menu click handles it. Run once per project (and again if you add new render features):

Tools > Runtime Lightbaker > Configure Project For Runtime Lightmaps.

It sets up everything the build needs:

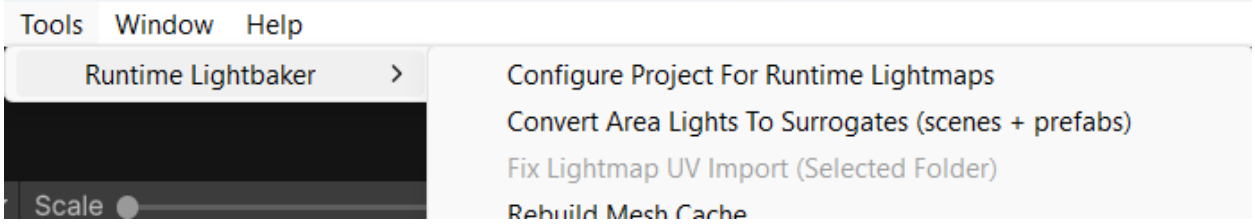
• Keeps the baker's shaders in your build so they aren't stripped out —the usual cause of a black/blank bake that only happens in a build.

• Turns OFF Static Batching for the build target (static batching rewrites mesh UVs and would corrupt the runtime lightmaps).

• Enables Realtime Reflection Probes so reflections actually refresh after a bake.

• On WebGL, switches on the fast lightmap path and a loader fallback so the build runs on any web host.

It only touches project-wide settings and prints a list of exactly what it changed — commit those to version control so your build matches the editor.



Also run once, after importing or changing models:
Tools > Runtime Lightbaker > Rebuild Mesh Cache — it makes big scenes load almost instantly and removes the need for Read/Write on your meshes

5. SCRIPTING / API (FOR PROGRAMMERS)

Everything runs from the Bake Orchestrator component. Grab a reference (GetComponent<BakeOrchestrator>) and call into it. Namespace: RuntimeLightbaker.Bake. All signatures below are exact.

5a. BAKING

• bool BakeAll(Action<bool> onCompleted = null)

Bake everything. With NO roots registered → the whole scene in one pass. With roots registered → each root as its OWN, independently unloadable lightmaps, one root at a time. Returns true if the bake started (or was queued behind another).

• bool BakeSelected(GameObject[] objects, Action<bool> onCompleted = null)

Re-bake only those objects and their (active) children into fresh slots; the rest of the scene keeps its lightmaps and still casts shadows / bounces light into the re-baked area. Returns false if the array is empty or none of the objects have active renderers.

• bool bakeOnStart = true

When true, BakeAll() is called automatically in Start(). Turn it OFF for streamed / spawned levels so YOU decide when the bake runs.

There is no Cancel method: calling BakeAll / BakeSelected again HARD- CANCELS the running bake and restarts it. (A deprecated Rebake() alias for BakeAll() still exists but emits a compiler warning — use BakeAll.)

5b. STREAMING LEVEL CHUNKS (spawn / bake / unload at runtime)

Why it matters: the "roots" API lets you bake and free level chunks one at a time as the player streams through the world.

• bool AddRoot(Transform chunk, bool bakeRoot)

Register a chunk parent as a bake root. With bakeRoot = true it also bakes JUST that chunk into its own lightmaps right now — and QUEUES behind any bake already running (it never cancels one). bakeRoot has no default, so always pass it. Returns true if the root was newly added (false on null or a duplicate).

• bool RemoveRoot(Transform chunk)

Unregister a chunk AND free the lightmaps it owns — atlas textures, slots, and renderer links. Every OTHER chunk is untouched (their lightmap indices never shift). Safe to call before OR after you Destroy the chunk. If a bake is in flight, the free is deferred until it settles.

• void ClearLightmaps()

Free ALL per-root chunk lightmaps at once — a full level teardown with no re-bake.

• IReadOnlyList<Transform> Roots { get; }

The registered roots (read-only; change it only via AddRoot / RemoveRoot). Empty means the next BakeAll bakes the whole scene.

Runtime chunk lifecycle (spawn → bake → unload):

1. Instantiate the chunk under ONE parent Transform (its renderers as children of that parent).
2. baker.AddRoot(chunkParent, bakeRoot: true); // register + bake it
3. (optional) wait for BakeCompleted, or poll CanRebakeNow, before adding the next chunk so passes don't pile up.
4. baker.RemoveRoot(chunkParent); // frees just this chunk
5. Destroy(chunkParent.gameObject);

```
  void OnChunkLoaded(GameObject c)   { baker.AddRoot(c.transform, true); }
  void OnChunkUnloaded(GameObject c) { baker.RemoveRoot(c.transform); Destroy(c); }
```

IMPORTANT — isolation. A chunk can only be freed on its own if it was baked as its own root: se AddRoot(chunk, true), or a BakeAll WITH roots registered. A whole-scene BakeAll (no roots) or a BakeSelected packs everything together, so RemoveRoot / ClearLightmaps CANNOT free those slots — they persist until the next full re-bake. Keep roots DISJOINT: an object placed under two roots is owned by whichever baked last.

5c. PROGRESS & STATE (for a loading UI or auto-rebaker)

• event Action<bool> BakeCompleted

Fires false when the fast baseline result is live, then true when the refined result is ready. A FAILED bake fires NEITHER — use IsDone to detect "finished, success or failure". (BakeAll / BakeSelected take the same onCompleted(bool) callback. In chunk mode, BakeAll's callback fires once after the last chunk, while the event fires per chunk.)

• float TotalBakeProgress { get; }

0..1 across baseline + refinement the best single value for a loader bar.

• float BaselineBakeProgress { get; }

0..1 through the fast first pass.

• float RefinementBakeProgress { get; }

0..1 through the background polish.

• bool IsDone { get; }

Reached its terminal state (Done OR Failed).

• bool CanRebakeNow { get; }

Safe to start a new bake without cutting off the final cleanup passes — the gate to prefer or automatic re-bakers.

• bool showLightmaps = true

Show / hide baked lighting instantly, no re-bake (display only — see Memory below).

5d. MEMORY MANAGEMENT

Why it matters: on long-running or streamed games you need to know what holds memory and how to give it back.

• **Baked lightmaps** (the atlas textures) are the main lasting cost. Reclaim them with:

- RemoveRoot(chunk) — frees just that chunk's lightmaps (streaming).
- ClearLightmaps() — frees ALL per-root chunk lightmaps (teardown).
- a full re-bake — replaces the previous set.

Only per-root lightmaps (AddRoot(_, true) or BakeAll-with-roots) can be freed this way; hole-scene / BakeSelected slots stay until a full re-bake.

• **showLightmaps = false** does NOT free memory. It only HIDES the lighting (unlinks the renderers); the textures stay resident. To actually reclaim VRAM use RemoveRoot / ClearLightmaps / a re-bake.

• **Bake-time scratch** — the scene's merged geometry + ray-tracing structure on the GPU, and the mesh cache in RAM — is freed AUTOMATICALLY when each bake finishes. You don't manage it; it's only resident DURING a bake.

• **bool ignoreVramBudget = true**

Leave ON in most cases (uses the full atlas budget). Set false to auto-cap the atlas COUNT to an estimated VRAM budget — safer against out-of- memory on tight devices, but it can leave far objects unlit if it caps too hard (watch the [RLBVRAM] console line).

• **bool useMeshCache = true**

Uses the pre-baked mesh cache (fast load, no Read/Write needed). Turning it OFF only falls back to slower runtime parsing — no correctness change.

Common leak to avoid: baking a chunk with a whole-scene BakeAll or a BakeSelected and then trying to unload it with RemoveRoot — those slots aren't per-root, so they won't free. Always bake unloadable chunks via AddRoot(chunk, true).

5e. COMPONENTS YOU CAN ADD IN CODE

For content you spawn at runtime, add these to the chunk BEFORE you bake it (all are plain AddComponent<> + set fields):

• **RLBEmissiveSurface** — mark a glowing mesh as a light source. Fields: emissionColor (HDR), emissionIntensity, boundsType, skipLightmap (default true), range metres; 0 = auto from size).

• **RLBAreaLight** — a bakeable area light. Fields: areaSize, shape, color, intensity, range, bounceIntensity, mode. (For authored scene lights prefer the Convert Area Lights tool — it also neutralizes the original Unity light so it doesn't leak in a build.)

• **BakeFlags** — per-object exclusion:

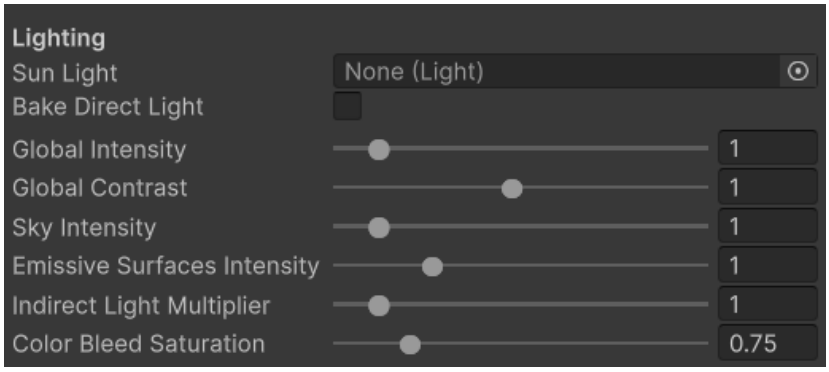
excludeFromBake (hide entirely), excludeFromShadowCasting (lit but casts no shadow), excludeFromTracing (lit but doesn't shadow or bounce — good for big backdrops).

(Don't add RendererLightmapOverrides yourself — it's generated at build time; set MeshRenderer.scaleInLightmap in the editor instead.)

6. FINE-TUNING (FOR POWER USERS)

Most people can skip this — the profiles already set these for you. This section is for when you want to fine-tune a scene or build a custom profile. The settings split across two places: the Bake Orchestrator component (per-scene) and the Bake Profile asset (reusable).

6a. ON THE BAKE ORCHESTRATOR (per-scene knobs)



BAKE DIRECT LIGHT.

Why it matters: whether the sun is baked in or stays live. Usually OFF. Baking the sun's shadows into the lightmap needs a very high Texels Per Unit to look sharp — and that simply can't be baked in seconds — so the sun normally stays realtime, casting crisp MOVING shadows on both baked and dynamic objects while the bake supplies the bounced light. Turn it ON only on mobile, when you need every last bit of runtime performance and soft / blurry sun shadows are acceptable.

GLOBAL / SKY / INDIRECT INTENSITY

• Global Intensity: overall brightness of everything.

• Sky Intensity: the outdoor/ambient light only - uses Unity’s “Environment Lighting” colors

• Indirect Light Multiplier: strength of bounced light only — raise it for a softer, fuller look in shadowed areas.

COLOR BLEED SATURATION.

Why it matters: how colorful the bounced light is. Higher = vivid and stylized (a red rug tinting the walls); lower = realistic. A great grounded-vs-stylized dial.

LIGHTMAP RESOLUTION (atlas size).

Why it matters: this is about PACKING, not sharpness (sharpness lives on the Profile — see Texels Per Unit). A bigger sheet fits more objects. Raise it only if huge surfaces look blurry or you're using too many sheets.

GATHER RANGE.

Why it matters: how far light travels and bleeds between surfaces. Set it to roughly your room / level size. Too small and light won't cross a room; too large and the bake costs more.

OPTIMIZATION BUDGETS (Min Trace Bounds Size, Max Lights, Max Emissive Surfaces).

Why it matters: your bake-time and runtime-cost safety valves.

• Min Trace Bounds Size skips tiny props (they use ambient instead) — **the single biggest speedup** in prop-heavy scenes.

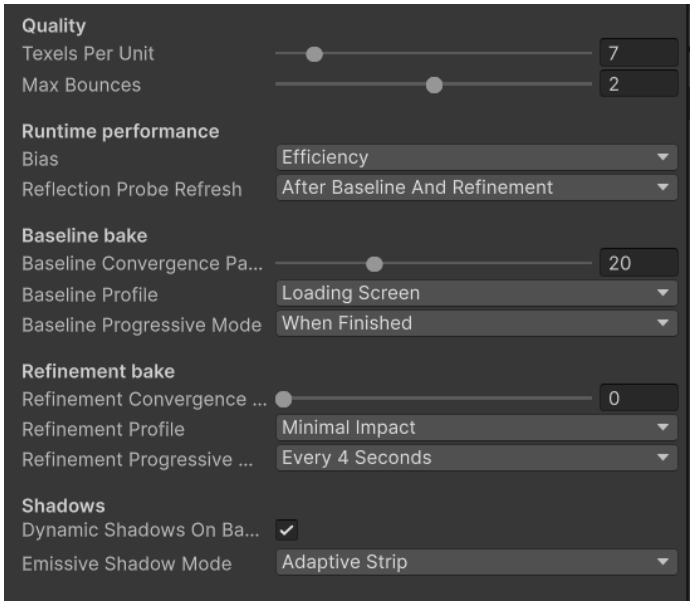
• Max Lights / Max Emissive Surfaces cap how many light sources the bake includes — lower them on weak hardware.

SHOW LIGHTMAPS.

Why it matters: flip your baked lighting on/off at runtime to A/B instantly.

6b. ON THE BAKE PROFILE (if you build your own)

A Bake Profile is a saved quality preset



TEXELS PER UNIT — THE **quality** dial.

Why it matters: the single biggest quality-vs-performance lever. Higher = crisper shadows and finer detail, but slower bakes and more memory. Distant cameras can go lower; surfaces the player gets close to want higher.

MAX BOUNCES.

Why it matters: how deep light reaches into shadow.

- 1 = only directly-lit surfaces glow (flat, cheap).
- 2 (default) = shadowed corners fill in naturally.
- 3+ = brighter enclosed interiors, slower bake.

Open scenes are fine at 1-2; enclosed interiors benefit from 2-3.

BASELINE vs REFINEMENT (passes + performance profile).

Why it matters: this is the two-phase system from section 3.

• Baseline Passes = the quick first look. Refinement Passes = the background polish (set to 0 for baseline-only, loading-screen-style bake).

• The Performance Profile decides how hard each phase pushes PER FRAME (i.e. your frame rate while baking): "Loading Screen" is fastest if a loading screen hides the game; "Background / Minimal Impact" keeps the game smooth while it finishes.

REFLECTION PROBE REFRESH.

Why it matters: when your shiny/reflective surfaces update to match the new lighting. Leave it on "After Baseline And Refinement" so reflections look right early and again at the end.

DYNAMIC SHADOWS ON BAKED.

Why it matters: after baking, turns the sun realtime so moving objects (and the sun) cast LIVE shadows over your baked lighting. The best-of-both-worlds default for any scene with moving things.

7. GOOD TO KNOW (FAQ)

Q: What if my model has no lightmap UVs (UV2)?

Why it matters: lightmaps need a second UV set; without one an object can't receive baked light — or bakes with overlapping, garbled lighting. The baker handles it automatically: by default it reuses your texture UVs when they work as a lightmap layout, and generates a fresh set when they don't. For the cleanest result, tick "Generate Lightmap UVs" in the model's import settings (or run **Tools > Runtime Lightbaker > Fix Lightmap UV Import on a folder of models)**. The "Missing Lightmap UV Handling" setting on the Bake Orchestrator lets you change this behaviour.

Q: Why are meshes cached? What does "Rebuild Mesh Cache" do?

Why it matters: it's the difference between a scene that lights up almost instantly and one that hitches for several seconds on load. The heavy part of baking is preparing each mesh's geometry for the light rays. The mesh cache does that work ONCE in the editor and stores the

result, so at runtime the baker just loads it — turning tens of seconds of setup into near-instant. Build it with **Tools > Runtime Lightbaker > Rebuild Mesh** Cache after importing or changing models. As a bonus, it also removes the Read/Write requirement (below).

Q: Do I need "Read/Write Enabled" on my meshes?

Why it matters: Read/Write roughly doubles a mesh's memory, so you don't want it on unless you have to. Usually NO. As long as the mesh cache is built (above), the baker reads geometry from the cache and never needs Read/Write. You'd only need it for meshes that AREN'T in the cache — for example meshes generated procedurally at runtime, or if you skip the cache entirely. Rule of thumb: build the mesh cache and leave Read/Write off.

Q: What if I have more than one Bake Orchestrator in a scene?

Why it matters: you might want one per streamed level chunk. That's fully supported. Multiple orchestrators bake ONE AT A TIME, in order (first-come-first-served) — never simultaneously — so they don't fight over the GPU. A single orchestrator with your chunks dropped into "Roots" is usually simpler, but multiple is fine when it fits your streaming setup.

Q: Where are the Light Probes?

Why it matters: if you're used to Unity, you might expect to place Light Probe Groups for your moving/dynamic objects. This baker doesn't use them, on purpose. Light probes are a FIXED grid you place and bake ahead of time. Because this baker runs LIVE inside your game —

where you might stream in or spawn level chunks at runtime — a pre-placed probe grid can't keep up and often ends up wrong or empty. So instead:

- Static objects get the full baked lighting (lightmaps).
- Moving / dynamic objects are lit by your realtime lights and the scene's ambient — and with "Dynamic Shadows On Baked" on, they cast and receive live shadows over the baked world.

In short: you skip the probe-placement chore, and moving objects still sit believably inside the lit scene.

FINAL CHECKLIST

[ ] Project Setup run once (Configure Project For Runtime Lightmaps)

[ ] Mesh Cache built (Rebuild Mesh Cache) — fast loading, no Read/Write

[ ] Bake Orchestrator on a GameObject, with a Profile assigned

[ ] The right profile family picked (LoadingScreen / SmoothFPS / Mobile)

at the quality tier your hardware can afford (HQ / MQ / LQ)

[ ] Roots set (or left empty for the whole scene)

[ ] Area lights converted to surrogates

[ ] Glowing objects given an RLB Emissive Surface

[ ] Press Play — and toggle Show Lightmaps to admire the before/after