

ReLayer User Guide

Unity 2022.3+ | React/Vite overlays | Version 1.1.7

An offline setup guide and API reference for building Unity HUDs, menus, live-ops panels, and overlays with ReLayer.

Published by Ryuxin Entertainment

Table of Contents

Table of Contents	2
1. About ReLayer	3
Included Workflows	3
2. Requirements and Compatibility	3
3. Install and First Run	4
ReLayer Studio	4
Files Created on First Setup	4
4. Project Structure and Ownership	5
5. Edit the Overlay	5
Recommended Edit Points	5
Editor Play Mode Sources	5
Recover an Unavailable Preview	6
Live Twin Browser Control	6
6. Build and Verify	6
WebGL DOM Overlay and Hosting	7
7. Player UI Updates	7
Hosted in Two Settings	8
Addressables and Persistent Installation	8
Custom Content Provider	9
Status, Recovery, and Trust	9
8. Unity and React Bridge	9
Unity to React	10
React to Unity	10
Typical Event Types	10
9. Input, Safe Area, and Scene Routing	10
Native Hit Testing	10
Safe Area	10
Scene Routing	10
10. Kits, Asset Broker, and Camera Render Broker	11
Reusable Kits	11
Asset Broker	11
Camera Render Broker	11
11. C# API Reference	11
12. Troubleshooting	12
13. Third-Party Notices	12

1. About ReLayer

ReLayer is a Unity UI workflow for editable React, Vite, TypeScript, HTML, and CSS overlays. It connects Unity and web UI through a structured C# <-> JavaScript bridge, and includes packaged WebView backends for supported players.

Use ReLayer for HUDs, menus, shops, inventory, settings, live-ops panels, forms, and full-screen in-game overlays.

Included Workflows

- A project-owned starter overlay for update-safe customization.
- ReLayer Studio: a packaged React editor workspace for delivery, diagnostics, builds, and offline guidance, with native Recovery always available.
- A packaged Showcase scene with components, themes, bridge examples, asset access, and camera render examples.
- A production build path that deploys browser assets through StreamingAssets.
- Optional Hosted, Addressables, and PersistentDataPath player UI delivery through the same ReLayer Initializer.
- Editable public Unity facades and samples, with first-party core implementations shipped as managed and native binaries.

2. Requirements and Compatibility

Item	Requirement or behavior
Unity	Unity 2022.3 or newer.
Package dependencies	No Unity Package Manager or Asset Store package dependency is required.
Node.js	Use Node.js ^20.19.0 or >=22.12.0 only to edit or rebuild the React/Vite overlay. Node.js 21 and 22.0-22.11 are unsupported. Fresh templates carry an npm lockfile; npm and pnpm approve only the reviewed esbuild@0.28.2 install step. Existing template source remains unchanged because ReLayer supplements only its isolated work copy.
Windows	Windows Editor Play Mode and Windows players are x86_64-only and require the Microsoft Edge WebView2 Runtime. Windows ARM64 is not supported.
Android	Uses the platform WebView at API level 21 or later. Packaged native ABIs are arm64-v8a and x86_64; ARMv7 and 32-bit x86 are not included.
macOS	Packaged WKWebView plug-in supports Apple Silicon arm64 only on macOS 11+. Intel, Rosetta/x64 Unity, and Universal player builds are not supported.
iOS	Packaged WKWebView XCFramework contains arm64 device and arm64 Simulator slices. The Simulator requires Apple Silicon; Intel/x86_64 Simulator is not supported.
WebGL	Uses a same-document browser DOM overlay. Production hosting requires HTTPS; plain HTTP is for loopback development only.
Linux	No first-party Linux backend is bundled. Linux is outside the supported runtime matrix.

The packaged Showcase runs without Node.js. The editable template and kits include a precompiled Tailwind CSS utility stylesheet; use normal CSS for new styles rather than installing a Tailwind compiler into the package workflow.

3. Install and First Run

On its first interactive import, ReLayer creates a project-owned starter scene and overlay workspace. It records a ProjectSettings marker so reimports and package updates do not repeat setup or overwrite your work.

1. Import ReLayer into a Unity 2022.3+ project and wait for Unity to finish importing Assets/ReLayer.
2. After successful setup, open Tools/ReLayer/Open Studio and choose Open Starter Scene. Alternatively, create an Initializer in your current scene.
3. Press Play. The starter scene includes a camera, light, and ReLayer Initializer. The default Auto source starts the project-owned Vite development server when Node.js is available.
4. Open Assets/ReLayer/Samples/Scenes/ReLayer_Showcase.unity to explore the packaged component and bridge examples.
5. Use Request Unity safe area on the starter page. The response panel shows normalized insets only after SAFE_AREA_CHANGED arrives from Unity; it does not infer success from a click. A plain browser shows a disconnected state, and an unanswered request has a timeout and retry path.

ReLayer Studio

Open Tools > ReLayer > Open Studio. Overview shows the next useful action; UI Delivery handles mode-specific settings, validation and builds; Diagnostics reports the game preview's readiness and failures; Guide contains offline instructions and references. The old UI Delivery and Preview Status menus remain shortcuts to these pages.

The target bar identifies the Initializer and actual settings owner. One target is automatic; multiple targets require a choice. Clicking another Unity object does not retarget your edits. Text commits on blur or Enter, invalid values remain unsaved, and Unity Undo/Redo applies to saved settings. Switching delivery modes preserves the other modes' values.

Studio opens its packaged React interface locally without internet, Node.js or your game development server. Game UI editing and rebuilding still require the normal toolchain. Closing Studio leaves a running build owned by Unity. Script reload interruptions are reported and never replayed automatically; inspect output before retrying. Studio's separate React runtime and Geist fonts include their notices in Assets/ReLayer/Core/Editor/Studio/Web/THIRD-PARTY-NOTICES.txt.

If Studio cannot start, use Tools > ReLayer > Recovery for native settings, validation, build and preview recovery. Studio requires a matching Editor-native payload. Older Apple payloads can use Recovery until the Studio-capable payload is installed and verified on Mac; existing Apple player delivery support is unchanged.

Files Created on First Setup

```
Assets/ReLayerUser/Scenes/ReLayer_Starter.unity
Assets/ReLayerUser/Overlay/Template
Assets/StreamingAssets/ReLayer/overlay
ProjectSettings/ReLayerFirstInstallAutomation.json
```

Do not edit Assets/ReLayerUser from a package update. ReLayer treats that folder as buyer-owned and preserves it after first setup.

4. Project Structure and Ownership

Path	Role	Edit?
Assets/ReLayerUser/Overlay/Template	Project-owned React/Vite source created on first run.	Yes
Assets/ReLayerUser/Overlay/Template/src/App.tsx	Main buyer overlay entry point.	Yes
Assets/ReLayer/Overlay/Template	Package-owned Showcase source.	No
Assets/ReLayer/Overlay/Kits	Package-owned public components and themes used through #relayer aliases.	No
Assets/StreamingAssets/ReLayer/overlay	Generated project overlay loaded by player scenes.	Regenerate
Assets/StreamingAssets/ReLayer/showcase	Built demo UI loaded by ReLayer_Showcase.	No
Assets/ReLayer/Documentation/ReLayer User Guide.pdf	This offline user guide and API reference.	Read

ReLayer omits the project overlay output from update packages, so package updates do not overwrite a generated game UI. The packaged Showcase output is separate and may be updated with the package.

5. Edit the Overlay

Customize the project-owned template, never package-owned files under Assets/ReLayer. ReLayer mirrors the active template into .relayer/profiles/<profileId>/OverlayTemplate for dependency installation and build compatibility work, leaving your source files unchanged.

Recommended Edit Points

src/App.tsx	HUD, menu, or app entry point
src/styles.css	Overlay styling
src/components	Project-owned reusable components
src/themes/custom	Project-owned theme presets

- Do not run npm install in Assets/ReLayerUser/Overlay/Template. ReLayer uses its isolated .relayer work copy for toolchain dependencies.
- Import supplied kits through #relayer/components, #relayer/themes, and #relayer/glass so package updates can refresh kit internals without rewriting your app source.
- Use src/relayer-fork only when you intentionally need a buyer-owned fork of kit internals. Forks do not receive later kit updates.
- Use Tools/ReLayer/Restart Overlay Session after changing the active overlay path or when restarting the local Vite session.

Editor Play Mode Sources

Source	Use
Auto	Default. Uses the managed Vite server and HMR for the project template. Falls back to a current built overlay when live startup is unavailable.
LiveDevServer	Explicitly uses the managed Vite server and HMR path.
BuiltOverlay	Loads current StreamingAssets output. Stale output requires an explicit rebuild; this selection does not switch to a live server.

Recover an Unavailable Preview

Open Studio > Diagnostics through Tools/ReLayer/Preview Status to see game-preview readiness, failure details and local logs. If Studio cannot load, Tools/ReLayer/Recovery provides native diagnostics without React. Automatic startup retries are limited to three attempts, 20 seconds apart, and wait through compilation or Play Mode transitions.

Automatic preview and fallback checks never install dependencies or build the app. They use existing current output or explain why it cannot be used. Freshness scans run in the background, with a 30-second cooldown after failure. Exit Play Mode before using Retry / Restart Preview or Tools/ReLayer/Rebuild Overlay. Rebuild installs missing work-copy dependencies and builds asynchronously, validating staged output before publication and preserving the last published output if the build fails. An automatic built fallback reconnects when its live server becomes ready.

Live Twin Browser Control

Choose Tools/ReLayer/Live Twin/Open to open an interactive browser twin of the actual Unity Play Mode WebView. ReLayer copies and logs the tokenized 127.0.0.1 URL and opens the default browser. The URL works with ordinary browsers and browser tooling in Codex, Claude, Cursor, and other editors without a product-specific extension.

- The Unity WebView remains the only authority. Live Twin does not run a second React app, reload the overlay, create another app bridge session, or inject input into the Unity Game View or 3D scene.
- Browser click, focus, text, selection-control, scroll, keyboard, pointer, hover, drag, and form commands execute on the existing WebView DOM. Up to eight browser tabs are serialized in server-arrival order and converge through the same authority patches.
- Vite development DOM exposes data-relayer-node-id, data-relayer-source, and data-relayer-component. Built overlays remain inspectable but report source metadata as unavailable when React debug evidence is absent.
- The schema-v1 window.__RELAYER_LIVE_TWIN_V1__ API keeps describeElement, inspectAuthority, and requestResync and adds getStatus, interact, click, focus, blur, setValue, setChecked, select, scroll, key, pointer, and submit. Targets are exact mirror elements or data-relayer-node-id values, never selectors or JavaScript source.
- Interaction Promises resolve after the authority result and settled revision reach that browser. A five-second timeout reports mayHaveExecuted: true and does not automatically retry a possibly non-idempotent action.
- DOM actions continue while Unity is paused, but React-to-Unity game work may wait for resume. Live Twin completion confirms DOM settlement, not game-logic completion.
- Local tools can discover the current URL and source root in .relayer/live-twin.json. The file never contains the encrypted ReLayer bridge key.
- Use Tools/ReLayer/Live Twin/Copy URL to copy the active address and Tools/ReLayer/Live Twin/Stop to close connections, release the port, and delete the discovery file.
- The server binds only to 127.0.0.1 and separates browser and authority tokens. The iframe remains script-free. Synthetic events cannot satisfy event.isTrusted or user activation; file selection/drop, clipboard permission, fullscreen, payment, download, external navigation, new windows, native form navigation, and arbitrary JavaScript are blocked.
- Browser rendering is not final pixel proof. The shell marks the attached twin INTERACTIVE and shows the live CSS viewport; the actual Unity WebView remains the rendering authority.

6. Build and Verify

Unity player builds rebuild the active overlay before Unity writes the player. Use the manual command when you want to inspect production output before a player build. For player UI delivery, Studio > UI Delivery > Build UI Update creates the mode-specific package and release manifest described in Chapter 7.

For a lighter build that excludes bundled example themes, add `-ExcludeExampleThemes`. The packaged Showcase remains isolated from your game build: it loads `Assets/StreamingAssets/ReLayer/showcase/index.html`, while user scenes load `Assets/StreamingAssets/ReLayer/overlay/index.html`.

1. Run the manual build or start a Unity player build.
2. Open `ReLayer_Showcase` and press Play to confirm that the built Showcase loads.
3. Open the starter scene, press Play, and confirm the overlay shows the current project-owned UI.
4. Build and run the intended player platform. Confirm platform prerequisites from Chapter 2 before investigating package behavior.

WebGL DOM Overlay and Hosting

WebGL uses a ReLayer-owned DOM layer in the same document as the Unity canvas, not an iframe. The preferred React mount is `#relayer-root`, while existing `#root` overlays remain supported. Transparent space stays available to Unity input; mark only interactive React surfaces with `data-relayer-hit`.

- Serve production players over HTTPS. Plain HTTP is intended only for localhost or 127.0.0.1 development, and `file://` is unsupported.
- Keep the player page, Build files, and `StreamingAssets/ReLayer` resources on one origin when possible. Split origins require CORS for every overlay resource.
- Allow Unity loader and WASM sources plus ReLayer `connect-src`, `script-src`, `style-src`, `asset` directives, and `blob:` module URLs in CSP. Provide a reusable nonce for inline import maps and scoped styles, or explicitly permit those inline blocks.
- ReLayer follows `Module.canvas` geometry across margins, resize, scroll, fullscreen, visual viewport, and canvas-size changes. Detached or zero-size canvases hide the overlay until valid geometry returns.
- Reload and destroy remove ReLayer-owned DOM, styles, scripts, Blob URLs, observers, listeners, and bridge state. Public C# bridge APIs are unchanged.

Compressed file	Content-Type	Content-Encoding
<code>.js.gz</code>	<code>application/javascript</code>	<code>gzip</code>
<code>.wasm.gz</code>	<code>application/wasm</code>	<code>gzip</code>
<code>.data.gz</code>	<code>application/octet-stream</code>	<code>gzip</code>

Validated runtime matrix: Unity 2022.3.62f2 WebGL IL2CPP with desktop Chrome and Edge, including source and exact-package builds, real Edge window resizing, canvas alignment, bridge round trips, and process cleanup. Unity 6, Firefox, Safari, and mobile browsers remain unverified.

7. Player UI Updates

Existing projects remain on `StreamingAssets`. Open Studio > UI Delivery, choose Bundled, Addressables, Hosted, Local, or Custom, run the tab's validation, then use its setup or build action. Switching tabs preserves every mode's values. Build UI Update creates a local immutable release; use Open output after completion. Users do not edit `release.json` or assign version numbers manually. Ship the loader in a player once before UI-only updates. Hot updates are not officially supported on WebGL; retain its existing web deployment.

Source	Delivery and update behavior
<code>StreamingAssets</code>	Default and backward-compatible. UI is included in the player and changes with a normal player build.

Source	Delivery and update behavior
Hosted	Loads an immutable release from one public HTTPS directory. Startup and 30-second foreground checks select a candidate without interrupting the current screen.
Addressables	Downloads a verified .bytes UI package through the project's existing catalog and installs it locally for offline reuse.
PersistentDataPath	Verifies and installs a generated release copied by an application-owned local patch process.
Custom	Receives one complete overlay.bytes archive from an application-owned IReLayerPlayerContentProvider.

Hosted in Two Settings

Run Validation checks setup and local package integrity. Failed setup blocks a build; damaged output can be rebuilt. After publishing, Test Hosted Release verifies HTTPS file hashes, MIME types, ETag/304 and pointer caching. Test rendering, the native bridge and application API CORS in Play Mode or on the target player.

1. Set UI source to Hosted and enter the public HTTPS deployment directory.
2. Press Build UI Update. Upload releases/<version>/ without changing its bytes, verify the remote files, then publish release.json last with no-cache and an ETag.
3. Let ReLayer detect the release. Detection never reloads the current screen, and background or focus recovery is not an apply point.
4. Hide and reopen the complete ReLayer view, or call ApplyPreparedPlayerOverlayAsync at a game-owned loading or lobby safe point.

```
await initializer.ApplyPreparedPlayerOverlayAsync();
```

Replacement stays hidden until the authenticated document boot, manifest version, React READY, and optional Unity state restoration succeed. The old document's delayed callbacks are fenced. Gameplay, audio, and purchase commands received during restoration are discarded instead of replayed. React memory is not preserved; Unity or the server remains authoritative.

Hosted falls back to the built-in StreamingAssets UI and does not guarantee an offline copy of the last remote website. Choose Addressables when the last installed remote UI must survive an offline restart.

Addressables and Persistent Installation

- Install an Editor-compatible Addressables release only for this mode. Windows-tested pairs: Unity 2022.3.62f2 / Addressables 1.21.19; Unity 6000.5.2f1 / Addressables 2.10.3. Addressables 1.21.19 cannot compile on Unity 6000.5. Keep the deployed player's toolchain and content-state baseline for UI updates.
- Set up UI Addressables creates or reuses only the ReLayer-owned group and key; it does not rewrite unrelated groups, remote URLs, or AWS credentials. Publish the generated catalog (.json or .bin), not the private content-state file.
- If the game owns catalog updates, choose Update control = Project and call InitializePlayerOverlayAsync once after that updater completes. ReLayer does not call UpdateCatalogs or duplicate the global catalog refresh.
- The first Addressables player build requires an explicit content-state baseline. Later UI updates use that deployed player's exact baseline and Unity's content-update workflow. Platform bundles and catalogs must be published by the project's existing CI.
- PersistentDataPath accepts a directory below Application.persistentDataPath. Both local modes reject traversal, links, duplicate or case-colliding paths, corrupt bytes, incompatible manifests, more than 10,000 files, or packages over 256 MiB before promotion.

Custom Content Provider

The Custom tab builds `overlay.bytes` and selects one explicit provider ID. Implement `IReLayerPlayerContentProvider` and register it before the `Initializer` starts. The provider receives profile, compatibility, and active-version context and returns the complete archive bytes. It cannot return a `WebView` URL or mark content trusted.

```
ReLayerPlayerContentProviders.Register("my-game.ui", provider);  
// provider.PrepareAsync(...) returns ReLayerPlayerContentPackage
```

ReLayer then performs the same `release.json` compatibility and SHA-256 checks, staging-directory installation, safe-point application, failed-version quarantine, built-in fallback, and last-good recovery used by `Addressables`.

Status, Recovery, and Trust

For a step-by-step S3 and CloudFront setup with actual English AWS Console screenshot cards, open `ReLayer-AWS-S3-Hot-Update-Guide.pdf` beside this guide. The concise source is `AwsS3HotUpdateTutorial.md`. Both use generic examples and keep credentials outside Unity and the UI output.

`InitializePlayerOverlayAsync` performs first preparation and display. `PreparePlayerOverlayAsync` checks, downloads, or installs without replacing the active document. `ApplyPreparedPlayerOverlayAsync` activates the prepared candidate. `PlayerOverlayStatusChanged` distinguishes checking, discovered, installing, installed, awaiting apply, applying, active, built-in, and failed states.

- Editor HMR remains the default. In Studio > UI Delivery, enable Test this delivery mode in Play Mode before entering Play Mode. Studio shows the active version, prepared version and available next action; a discovered Hosted candidate is not an installed download.
- Advanced startup flows can use `ShowBuiltInPlayerOverlayAsync` for the packaged shell, `SetPlayerOverlayAddressableKey` before delivery starts, and `ActivateLastGoodPlayerOverlayAsync` for an already verified local UI. No global catalog is refreshed by these calls.
- `RestorePlayerOverlayStateAsync` resends authoritative snapshots before display. Inside that hook, `PlayerOverlayRestoreVersion` identifies the document being restored, including rollback or the built-in shell; use it to select matching state and media roots, not to replay commands.
- `ReLayerContentPackInstaller.InstallAddressableAsync` verifies separately bundled media against its manifest, hashes and immutable identity. Use the returned `ReLayerInstalledContentPack` URLs; do not expose unverified downloaded files directly.
- Hosted accepts only the configured HTTPS origin and deployment directory. Redirects cannot widen that boundary, and external frames, popups, and pages do not receive the Unity bridge.
- Every Hosted document has a fresh authentication value even when the legacy bridge security setting is `PlainText`. Generated CSP blocks frames, objects, forms, and external scripts; extra resource origins are explicit advanced settings, not trusted pages.
- A failed activation is quarantined until a different version is published or Retry failed version is selected. Local modes first try their last verified installation; Hosted uses the built-in UI.
- Use `Assets/ReLayer/Core/Build/verify-ui-update.py` in CI before and after Hosted publication. The verifier is read-only and never uploads files or handles cloud credentials.
- Player UI delivery changes HTML, JavaScript, CSS, and referenced resources only. C#, native functionality, and store-policy-sensitive behavior still require the appropriate player release and review.

8. Unity and React Bridge

Use the structured bridge for messages between Unity and the overlay. Unity code sends typed events through `ReLayerBridgeManager`; React sends messages through `bridge.send(type, payload)`.

Unity to React

```
ReLayerBridgeManager.GetOrCreate().Send(  
    ReLayerBridgeEventType.UnityToUiToast,  
    new ReLayerToastPayload  
    {  
        title = "Unity event received",  
        body = "Sent from Unity."  
    });  
  
var bridge = ReLayerBridgeManager.GetOrCreate();  
bridge.On(ReLayerBridgeEventType.UiAction, payload => Debug.Log(payload));
```

React to Unity

```
bridge.send('UI_ACTION', { action: 'pause' })
```

Use `hasUnityRuntimeConnection()` for Play-only loading gates and commands. `hasUnityRuntimeBridge()` only confirms that a native WebView callback exists; it can be true during an Edit Mode preview.

Typical Event Types

Direction	Events
Unity to React	READY, NAVIGATE_TO, SCENE_CHANGED, SCENE_ROUTE_MATCHED, PLAYER_STATE_SYNC, UNITY_TO_UI_TOAST, ASSET_IMAGE_READY, CAMERA_RENDER_READY
React to Unity	UI_ACTION, BUTTON_CLICK, OPEN_EXTERNAL_URL, ASSET_PREFETCH, ASSET_IMAGE_REQUEST, ASSET_AUDIO_PLAY, CAMERA_RENDER_REQUEST, CAMERA_RENDER_RELEASE

9. Input, Safe Area, and Scene Routing

Native Hit Testing

Add `data-relayer-hit` to every clickable React surface. ReLayer reports those rectangles to the native overlay so controls receive WebView input while transparent overlay areas remain click-through to Unity during Play Mode. Render In Edit Mode is intentionally a non-interactive visual preview.

```
<button data-relayer-hit  
    onClick={() => bridge.send('UI_ACTION', { action: 'pause' })}>  
    Pause  
</button>
```

Safe Area

`SAFE_AREA_CHANGED` provides normalized left, top, right, and bottom insets after the WebView is ready, after display changes, and when React asks for a refresh. Existing overlays can ignore this optional event.

```
const stopSafeArea = subscribeSafeArea(({ top }) => {  
    document.documentElement.style.setProperty(  
        '--safe-top', `${top * window.innerHeight}px`  
    )  
})  
requestSafeArea()  
// Call stopSafeArea() when the component unmounts.
```

Scene Routing

Open Scene options on ReLayer Initializer, enable Scene Routing, and map each Unity scene to a route, for example IntroScene to `app-start` and GameScene to `battle`. Lifecycle options are in the same foldout. When a legacy settings asset is linked, shared options edit that asset; scene routes stay on the initializer. Do not add a second `ReLayerBridgeManager` object manually.

10. Kits, Asset Broker, and Camera Render Broker

Reusable Kits

```
import { Button, Card, TextField } from '#relayer/components'
import { GlassButton, GlassPanel, GlassRoot } from '#relayer/glass'
import { getComponentTheme } from '#relayer/themes'
```

Asset Broker

Allowlist public asset IDs in Unity. React should request only those IDs, never project paths or Addressables keys.

```
const image = useReLayerAssetImage('showcase.badge.icon')
image.request()
```

Camera Render Broker

The scene initializer owns the Camera Render Broker. Allowlist camera targets before React can request a preview.

```
const preview = useReLayerCameraRender('main-camera')
preview.request({ width: 320, height: 180, transparent: false })
preview.release()
```

11. C# API Reference

These public facades are readable and intended for scene configuration and integration. Session security, local resource serving, Addressables lifecycle, and platform backend implementation remain in ReLayer first-party managed DLLs and native plug-ins.

Type	Purpose
ReLayerInitializer	Creates bridge, overlay settings, and WebView integration for a scene.
ReLayerPlayerOverlayOptions	Selects Bundled, Hosted, Addressables, PersistentDataPath, or Custom delivery and its advanced settings.
IRelayPlayerContentProvider	Supplies a complete custom UI archive while ReLayer retains validation, installation, safe application, and rollback.
ReLayerPlayerOverlayStatus	Reports preparation, activation, active version, prepared version, and recovery state.
ReLayerBridgeManager	Sends and receives structured C# to JavaScript bridge events.
ReLayerUnityToReactEvent	Inspector-friendly UnityEvent relay from Unity to the overlay.
ReLayerReactToUnityEvent	Inspector-friendly listener for overlay-originated events.
ReLayerSceneOverlayRouter	Maps Unity scene changes to overlay routes.
ReLayerOverlayWebView	Manages backend selection, loading, visibility, and overlay lifecycle.

12. Troubleshooting

Symptom	Action
Live overlay does not start	Install a supported Node.js version and set RELAYER_NPM_PATH to its npm or pnpm. If Windows reports a non-ASCII work-path failure, select a current Node.js 24 runtime, then run Tools/ReLayer/Restart Overlay Session.
Need a runnable demo without Node.js	Open ReLayer_Showcase. It is fixed to BuiltOverlay and loads the packaged Showcase output.
Want to inspect production output	Select BuiltOverlay or run Assets/ReLayer/Core/Build/build-overlay.ps1.
Hosted update is detected but not visible	This is expected until a safe point. Hide and reopen the full ReLayer view, or call ApplyPreparedPlayerOverlayAsync from a game-owned loading or lobby boundary.
Hosted is unavailable offline	Hosted guarantees the built-in UI fallback, not the last remote site. Use Addressables when a verified installed update must remain available offline.
Addressables update does not appear	Finish the game's catalog update first, then prepare and apply ReLayer. Confirm the build used the deployed player's exact content-state baseline and correct platform profile.
Custom provider is not registered	Register the exact Provider ID before the Initializer starts. Return the generated overlay.bytes archive, not an entry URL or unpacked files.
Live Twin shows Waiting	Enter Play Mode and confirm one ReLayer WebView is active. The browser attaches without reloading the overlay.
Live Twin action timed out	Do not repeat a click blindly. A timeout reports mayHaveExecuted because the authoritative DOM may already have handled it; request a resync and inspect the current revision first.
Live Twin DOM changed but Unity did not react while paused	Resume Unity. Live Twin confirms authoritative WebView DOM settlement; React-to-Unity game bridge processing can remain queued during pause.
React controls do not receive input	Add data-relayer-hit to the clickable surface and test in Play Mode, not Render In Edit Mode.
WebGL is blank or reports a module/style load error	Use production HTTPS, verify same-origin or CORS access, and allow the required CSP origins, blob: modules, and inline nonce policy.
Compressed WebGL files fail to load	Serve .js.gz as application/javascript, .wasm.gz as application/wasm, and .data.gz as application/octet-stream, all with Content-Encoding: gzip.
A package update overwrote UI work	Move project edits to Assets/ReLayerUser/Overlay/Template. Assets/ReLayer is package-owned and can be replaced by reimport.
Windows backend is unavailable	Install the Microsoft Edge WebView2 Runtime, then restart Unity.

For sample content, start with Assets/ReLayer/Samples/Scenes/ReLayer_Showcase.unity. For support, use <https://discord.gg/b5rAw8eYJu>.

13. Third-Party Notices

This chapter summarizes third-party components used by the editable overlay. ReLayer itself is licensed only under the Unity Asset Store End User License Agreement; the MIT and ISC texts in Third-Party Notices.txt apply only to the listed third-party components.

ReLayer Third-Party Notices

1. Scope and ReLayer License

This file identifies the third-party components redistributed with the ReLayer Unity Asset Store package under Assets/ReLayer and Assets/StreamingAssets/ReLayer.

ReLayer and all first-party ReLayer source, binaries, documentation, samples, and artwork are licensed only under the Unity Asset Store End User License Agreement. Nothing in this file grants, restricts, replaces, or adds a license for ReLayer itself. The MIT and ISC license texts below apply only to the third-party components identified in Section 4.

The package does not redistribute node_modules, npm cache files, source trees for third-party native WebView implementations, or third-party Unity packages. The editable overlay template includes package.json so a buyer can install the optional web build toolchain in a project-local work folder. The package includes a precompiled Tailwind CSS utility stylesheet at Assets/ReLayer/Overlay/Template/src/styles/tailwind-utilities.css; it does not install a Tailwind compiler into buyer projects.

2. First-Party Package Content

ReLayer C# source, sample assets, managed DLLs, and native plug-ins are first-party Ryuxin Entertainment / ReLayer artifacts unless this notice says otherwise. This includes:

- Assets/ReLayer/Core/Runtime/Plugins/Managed/ReLayer.Core.dll
- Assets/ReLayer/Core/Runtime/Plugins/Managed/ReLayer.NativeBridge.dll
- Assets/ReLayer/Core/Runtime/Plugins/Windows/x86_64/ReLayerNative-<version>.dll
- Assets/ReLayer/Core/Runtime/Plugins/Android/ReLayerNative.aar
- Assets/ReLayer/Core/Runtime/Plugins/iOS/ReLayerNative.xcframework
- Assets/ReLayer/Core/Runtime/Plugins/macOS/ReLayerNative.bundle
- Assets/ReLayer/Core/Runtime/Plugins/WebGL/ReLayerWebGL.jslib

Unity-serialized facades and sample scripts remain readable for scene setup and supported integration. Session security, local resource serving, Addressables lifecycle, and platform backend implementation are supplied in the first-party managed DLLs and native plug-ins; their original publisher source is not redistributed in this package.

3. Platform Requirements

No third-party managed or native WebView package is bundled. Windows Editor Play Mode and Windows players are x86_64-only and require the Microsoft Edge WebView2 Runtime. Android uses the platform WebView at API level 21 or later; the packaged native ABIs are arm64-v8a and x86_64. macOS uses WKWebView on Apple Silicon arm64 only. iOS device and Simulator slices are arm64; the Simulator slice requires an Apple Silicon host. WebGL uses the browser DOM. No first-party Linux backend is bundled. These platform services are not redistributed by ReLayer.

4. Web Overlay Components

The packaged Showcase and Starter overlay output is generated from the editable React/Vite template. The following direct components are used by that template or its generated output:

Component	Version	License	Use
React	19.2.7	MIT	Overlay UI runtime
React DOM	19.2.7	MIT	Overlay DOM rendering
Scheduler	0.27.0	MIT	React scheduling dependency
clsx	2.1.1	MIT	Class-name composition
tailwind-merge	3.6.0	MIT	Tailwind class merging
lucide-react	0.468.0	ISC	Template and Showcase icons
Vite	7.3.6	MIT	Optional local dev server/build

Component	Version	License	Use
Tailwind CSS	4.3.1	MIT	Precompiled template utility stylesheet
nanoid	3.3.18	MIT	Pinned Vite/PostCSS transitive dependency

5. Copyright Notices

React, React DOM, Scheduler Copyright (c) Meta Platforms, Inc. and affiliates.

Vite Copyright (c) 2019-present, VoidZero Inc. and Vite contributors.

Tailwind CSS Copyright (c) Tailwind Labs, Inc.

clsx Copyright (c) Luke Edwards <luke.edwards05@gmail.com> (lukeed.com)

tailwind-merge Copyright (c) 2021 Dany Castillo.

nanoid Copyright 2017 Andrey Sitnik <andrey@sitnik.ru>

lucide-react Copyright (c) for portions of Lucide are held by Cole Bemis 2013-2022 as part of Feather (MIT). All other copyright (c) for Lucide are held by Lucide Contributors 2022.

6. MIT License Text for Listed Third-Party Components Only

This text applies only to the components marked MIT in Section 4. It does not apply to ReLayer or to any first-party ReLayer file.

MIT License

Permission is hereby granted, free of charge, to any person obtaining a copy of this software and associated documentation files (the "Software"), to deal in the Software without restriction, including without limitation the rights to use, copy, modify, merge, publish, distribute, sublicense, and/or sell copies of the Software, and to permit persons to whom the Software is furnished to do so, subject to the following conditions:

The above copyright notice and this permission notice shall be included in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.

7. ISC License Text for lucide-react Only

This text applies only to lucide-react in Section 4. It does not apply to ReLayer or to any first-party ReLayer file.

Permission to use, copy, modify, and/or distribute this software for any purpose with or without fee is hereby granted, provided that the above copyright notice and this permission notice appear in all copies.

THE SOFTWARE IS PROVIDED "AS IS" AND THE AUTHOR DISCLAIMS ALL WARRANTIES WITH REGARD TO THIS SOFTWARE INCLUDING ALL IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS. IN NO EVENT SHALL THE AUTHOR BE LIABLE FOR ANY SPECIAL, DIRECT, INDIRECT, OR CONSEQUENTIAL DAMAGES OR ANY DAMAGES WHATSOEVER RESULTING FROM LOSS OF USE, DATA OR PROFITS, WHETHER IN AN ACTION OF CONTRACT, NEGLIGENCE OR OTHER TORTIOUS ACTION, ARISING OUT OF OR IN CONNECTION WITH THE USE OR PERFORMANCE OF THIS SOFTWARE.