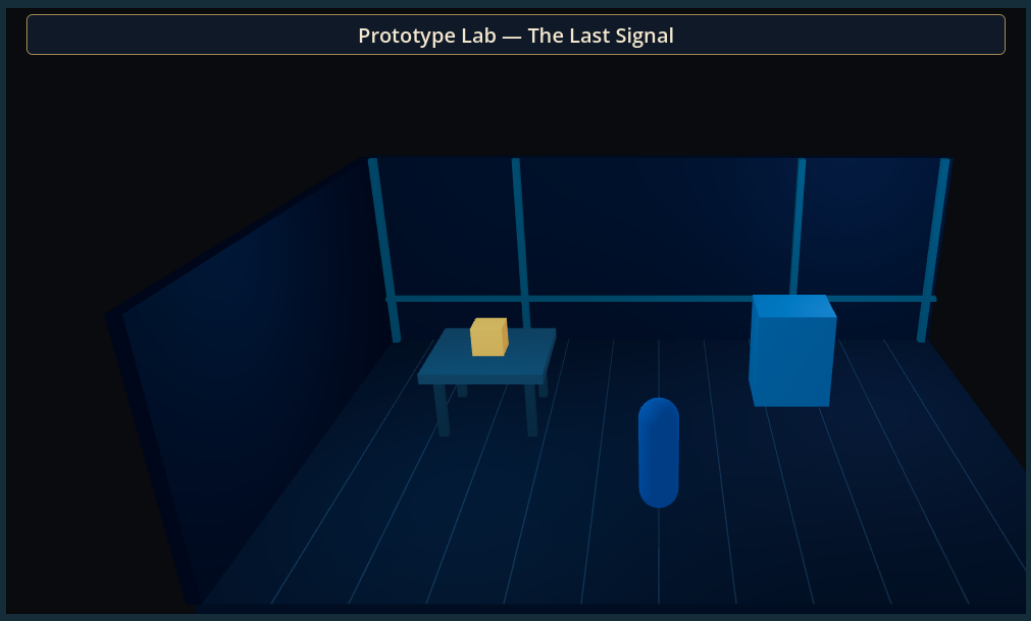
PCA 3D ADVENTURE STUDIO

Build Tools and Game Export

Take an authored PCA3D game from the editor to a folder you can actually run, send to a tester, or host on the web.



The same game can become a desktop package, an editable Godot project, an Android app, an iOS app or a browser game. Start with what your player needs to open.

Illustrated with PCA3D 0.86.3.4 on Linux

CONTENTS

Find a topic

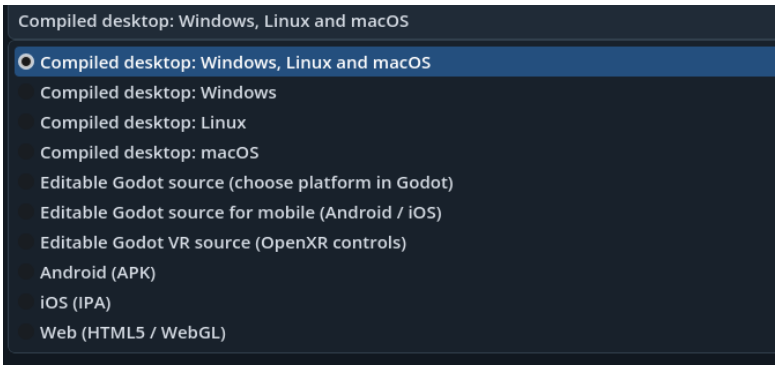
For Windows, Linux or macOS, start with the desktop walkthrough. For a phone or browser, read the Build Tools setup before your target’s chapter. Select a title to jump there.

Choose the right output	3
Set the identity players will see	4
Make a compiled desktop build	5
Know which files to send	6
Test the game outside the editor	7
Export editable Godot source	8
Set up the Build Tools dialog	9
Pick the computer that does the work	10
Build an Android APK	11
Build an iOS IPA	12
Prepare a signed mobile release	13
Build and host the web version	14
Set up a private remote build worker	15
Submit a job and collect its result	16
Check media, scripts and rendering	17
Read the first useful build error	18
Know when the build is ready to share	19

BUILD GAME

Choose the right output

Build Game makes a game from the project you have open. Its target selector answers a practical question: what do you want to hand to the next person?



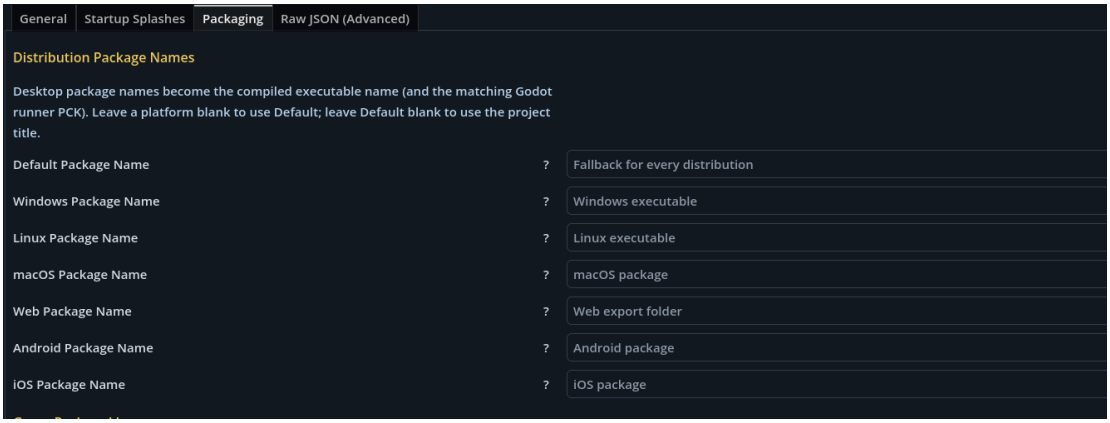
The open target menu includes four compiled desktop choices, three editable source choices and three mobile or web build targets.

A desktop player	Select Compiled desktop for Windows, Linux, macOS, or all three. PCA3D packages the matching bundled runner with the game data.
A developer using Godot	Choose Editable Godot source. The standard, mobile and VR variants produce a project the recipient can open and change.
An Android tester	Choose Android (APK) for an installable ARM64 app. An Android App Bundle for a store follows the editable source route.
An iPhone or iPad tester	Choose iOS (IPA). Compilation and signing run on a Mac equipped for iOS builds.
A browser player	Choose Web (HTML5 / WebGL). The result is a folder of web files that must be hosted together.

PROJECT SETTINGS → PACKAGING

Set the identity players will see

The game title, the desktop executable name, the game data pack and a mobile bundle identifier have different jobs. Set the ones that need to stay stable before release.



The Packaging tab has a default name and separate platform overrides. The game pack and icon choices sit below this group.

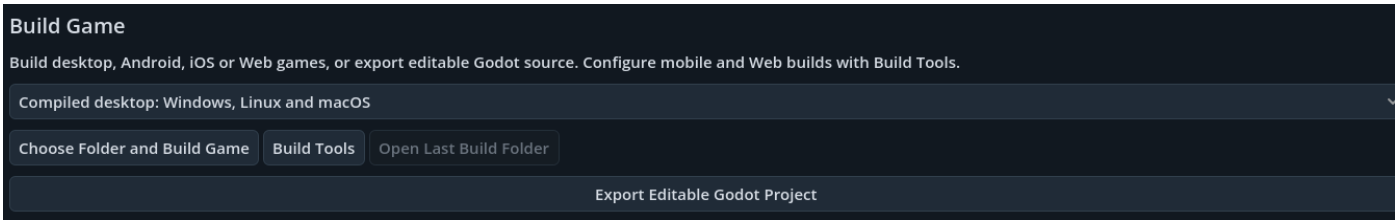
Package names	Enter a default package name and override it where needed. A blank platform field falls back to Default. If Default is blank, the builder derives a filename from the game title.
Game PCK Filename	PCK means a Godot data pack. A desktop build has a pack matching the executable name and a separate game-content pack. Choose a distinct game pack name. The default game pack in the checked build was game.pck.
Game Icon Image	Select an imported image asset. On Linux the branded build included game-icon.png. For a custom Windows executable icon on a non-Windows host, this build invokes its Windows icon helper through Wine.

The test game was titled Beacon Station, so its Linux executable was beacon-station.x86_64. The title displayed inside the sample room came from an authored HUD label. Change that label separately if the title inside your game looks out of date.

DESKTOP WALKTHROUGH

Make a compiled desktop build

For a first test, build for the computer in front of you. It gives you the shortest path from selecting a target to finding out whether the game starts.



The Build Game workspace. Choose a target above Choose Folder and Build Game. Build Tools opens the separate mobile and web settings.

01 Apply pending designer edits and save your project. Run **Validate** and resolve blocking errors. In the main toolbar select **Build Game**.

02 Select **Compiled desktop: Linux**, **Windows** or **macOS**. The all-three option packages each platform in its own folder. Mobile signing and the Build Tools Godot path are unnecessary for this desktop route.

03 Select **Choose Folder and Build Game**. Choose a writable parent folder with room for the build. PCA3D validates the project and creates a named child folder. Repeated builds use distinct folder names.

04 Read the result dialog and use **Open Last Build Folder**. Open the platform subfolder. On Windows launch the .exe, on Linux the .x86_64 executable, and on macOS the .app bundle.

> In the checked Linux-only build, the editor reported “Compiled desktop game created.” A separate all-platform run produced windows, linux and macos directories. A completed package still needs a playtest on the intended operating system.

DESKTOP DISTRIBUTION

Know which files to send

Launching the game from its fresh build folder is easy. The common packaging mistake comes afterward: someone sends the executable and leaves the game behind.

beacon-station.x86_64	The Linux executable from the checked build. Windows produced beacon-station.exe. The Mac package was beacon-station.app.
beacon-station.pck	The runner’s matching data pack. Its base name follows the desktop executable.
game.pck	The authored game content pack. pca3d-launch.json tells the runner which game pack to load.
override.cfg and libraries	The package also contains renderer configuration, native libraries and any required addon directories. Keep their relative locations intact.
build_info.json	The parent build folder records such details as the packaged platforms and runtime version. It can help identify the build you sent.

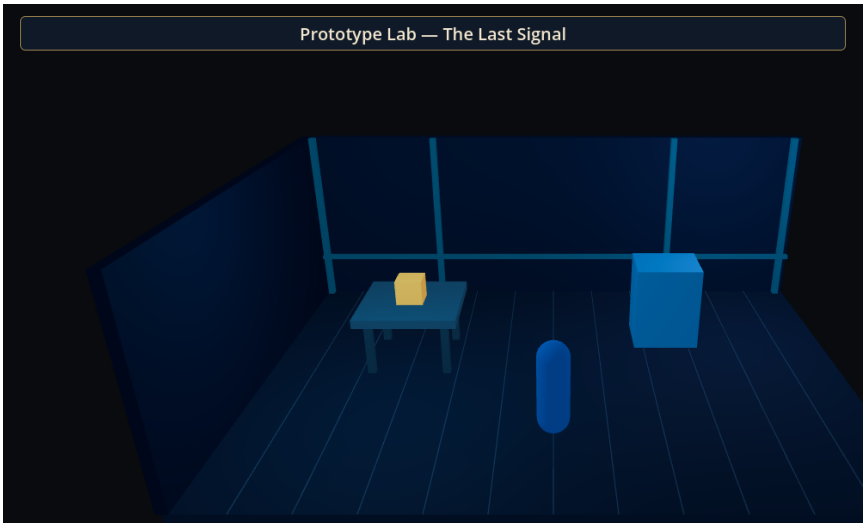
For one Windows or Linux tester, zip that complete platform folder. For macOS, preserve the whole .app bundle and its internal executable permissions. The desktop builder reports its Mac app as unsigned. Plan signing and notarization for public Mac distribution.

> Extract the delivery into another folder before testing it. On Linux also check that the executable permission survived the transfer. A run from the original folder cannot reveal that a file was left out of the ZIP.

DESKTOP VERIFICATION

Test the game outside the editor

A successful build dialog confirms that packaging finished. A launch from the platform folder tells you whether that package can find its runtime and game data.



Beacon Station running from the packaged Linux executable. The room and its authored HUD loaded outside PCA3D.

01 Launch from the copy you intend to distribute. Check the expected start room, splash or menu, authored artwork and audio. Record the exact build folder used.

02 Use the controls a player will use. Move, use a hotspot, open a GUI and make a room transition. Save into a test slot, close the game, then reopen it and load that slot.

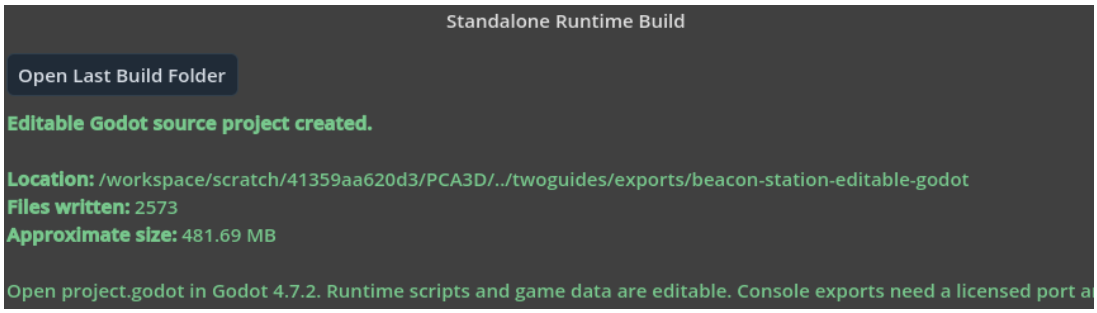
03 For an all-platform package, repeat on each supported OS. Windows and macOS binaries were packaged for this guide. Linux was the platform where the standalone game was launched and the room observed.

A custom Windows icon initially stopped an all-platform build on Linux because Wine was unavailable. Keeping the test project’s default icon allowed all three packages to finish. A branded Linux-only build also completed with the original icon selected.

SOURCE EXPORT

Export editable Godot source

Choose this route if you or another developer needs to inspect the runtime, customize Godot’s export presets, or finish a target outside PCA3D.



The source export result identifies an editable project. Open project.godot in the generated folder.

01 In **Build Game**, select **Editable Godot source (choose platform in Godot)** and choose a parent folder. The dedicated **Export Editable Godot Project** button selects this standard source route directly.

02 Open the generated **project.godot** in the compatible Godot editor. In this release the expected editor is **4.7.2**. The source output includes readable runtime scripts, **game/project.pca3d** and **export_presets.cfg**.

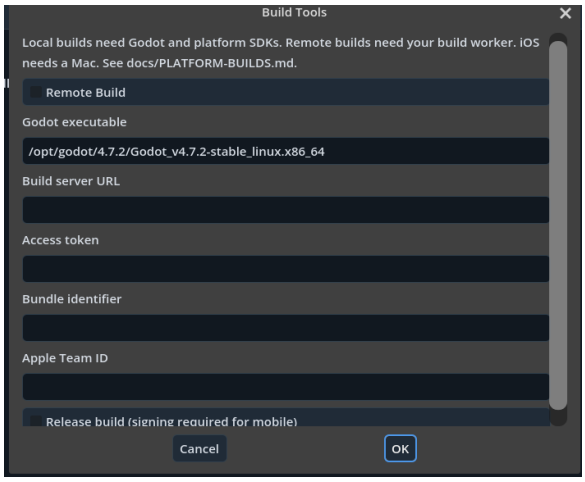
03 Run the generated game, then open **Project → Export** in Godot. Select the preset and configure its templates, SDKs and signing for the target. A source directory is an intermediate deliverable until you actually export a player build.

> Standard, mobile and VR source variants were generated and inspected. The mobile variant enabled mouse emulation from touch. The VR variant enabled OpenXR and the game’s VR setting. Test controller bindings and headset behavior on the device before offering a VR build.

ANDROID, IOS AND WEB

Set up the Build Tools dialog

Build Tools tells PCA3D which computer does the compilation. Start with the local route if this desktop already has the right Godot editor and platform tools installed.



Build Tools lets you choose local or remote compilation and configure the bundle identifier, Apple team and release mode.

Remote Build off	Enter the full path to the Godot 4.7.2 editor executable on this computer. On macOS this usually means the executable inside Godot.app/Contents/MacOS. PCA3D checks the version.
Remote Build on	Enter the worker's server URL and access token. The worker supplies the compiler, templates and platform SDK. The token is held for this editor session and must be entered after restarting.
Bundle identifier / Apple Team ID	Use a stable mobile app identifier such as com.yourstudio.beaconstation. For iOS, supply the Apple team used by signing unless the worker's profile already sets it.
Release build	Leave it off for a debug/test export. Enable it after the selected build computer has working release-signing credentials for the mobile target.

HOST AND TARGET

Pick the computer that does the work

The machine running PCA3D and the machine compiling the game can be different. Installing a mobile SDK on your phone does not configure a Mac or desktop worker.

Windows or Linux desktop	Compiled desktop uses the bundled runners. Android and Web may use the local Godot toolchain or a remote worker. Send iOS jobs to a Mac with Xcode.
macOS desktop	Android and Web can build locally with their dependencies, or remotely. iOS can run locally when Godot, Xcode and signing are configured, or on another configured Mac.
Phone, tablet or browser editor	For the APK, IPA and Web targets, the in-app builder directs these hosts to Remote Build. The client must be able to reach the worker and stay open while the request runs.
Computer doing the build	Install the full Godot editor and its matching export templates under the same account used to compile. Install each target’s platform tools on that machine. A mismatched template version can block export even when the project imports correctly.

Which setting belongs where?

Your game’s title, icon and package names belong in the project. Build Tools options belong to the user and device. Signing keys and provisioning files belong on the machine that performs the signed export. Switching workers means verifying the new machine’s setup.

ANDROID

Build an Android APK

An APK is an installable Android app. The in-app route in this release creates an ARM64 APK, which makes it a reasonable first artifact for testing on a compatible phone.

01 On the build computer, install Godot 4.7.2 and matching Android export templates, OpenJDK 17 and the Android SDK. The PCA3D release notes cite Android platform 35, platform-tools and build-tools 35.0.1 as a tested setup.

02 In the full Godot editor, open **Editor Settings → Export → Android**. Set Java SDK Path and Android SDK Path. Configure a debug keystore for development. Run Godot as the same OS user that PCA3D or the worker will use.

03 In PCA3D choose **Android (APK)**. Open Build Tools, select local or remote, set the bundle identifier if needed, leave Release build off for the first test and confirm the dialog. Then choose the output parent and build.

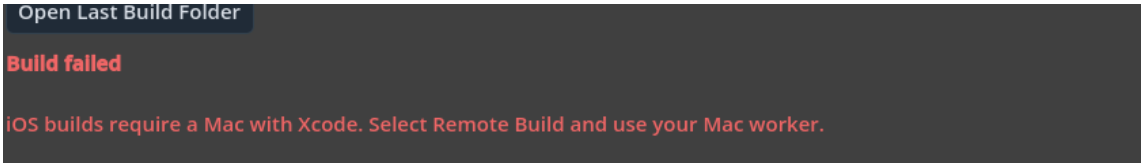
04 For a successful local export, look under **build/android/game.apk** in the generated project. For a remote build, extract the returned archive. Install the APK on an ARM64 device you control and test touch input, UI scale, audio and save/load.

> For release signing, configure the release keystore path, alias and password on the build machine. For a store workflow requiring an Android App Bundle (.aab), use editable mobile source and configure Godot’s Gradle/AAB export. Renaming an APK does not convert it to an AAB.

IOS

Build an iOS IPA

An IPA is a packaged iOS app. A Windows or Linux machine can prepare mobile source or submit a remote build. A Mac with Xcode performs the iOS compilation and signing.



The Linux editor stopped a local iOS build and directed it to a Mac worker. A new output folder would not change the host requirement.

01 On the Mac, install a compatible Xcode with its iOS SDK, complete its initial setup and license acceptance, then install the matching Godot editor and iOS templates.

02 Make the bundle identifier, Apple Team ID, certificate and provisioning profile agree. Select signing appropriate to development or distribution. Check that the account running the build can access the certificate and profile.

03 Select **iOS (IPA)**. On a Mac choose local or remote. From another desktop or mobile client, select Remote Build and enter the Mac worker URL and token. Choose a writable output parent and start the build.

04 A successful local job should produce **build/ios/game.ipa**. The remote result should contain **game.ipa**. Install using the route allowed by that signing profile and launch on a supported iOS device.

If you need to inspect the native project, use **Editable Godot source for mobile** and finish the export in Godot and Xcode. Creating an IPA does not send it to App Store Connect.

BUILD TOOLS · RELEASE BUILD

Prepare a signed mobile release

The Release build switch asks Godot for a release export. A working debug build is a good starting point, but the release identity still has to be configured on the machine doing the signing.

Android keystore	The keystore file holds the signing key. The alias identifies the key inside it. Configure the release path, alias and password for the Godot process on the build computer.
Android setting names	The documented variables are GODOT_ANDROID_KEYSTORE_RELEASE_PATH, GODOT_ANDROID_KEYSTORE_RELEASE_USER and GODOT_ANDROID_KEYSTORE_RELEASE_PASSWORD. USER here is the key alias. Use the values for your game’s own keystore.
iOS signing	Use the bundle ID registered for the app, the matching Apple team, a signing identity and a provisioning profile appropriate to the distribution method. On a remote Mac, those resources must be usable by the worker account.
Test the release artifact	Rebuild with Release build enabled, retrieve the new file and verify it through the intended installation or distribution path. Keep that artifact separate from the earlier debug export.

If a signed update is rejected, compare it with the earlier release. Check the app identifier and signing identity before editing gameplay. Preserve signing credentials privately so a future update can use the correct identity.

WEB / HTML5

Build and host the web version

A browser game consists of several files that must be served together. Opening a lone index.html from a file manager is a poor test of a web export.

01 Choose **Web (HTML5 / WebGL)**. Set the path to the compatible local Godot editor, or select a configured Web worker. Install matching Web export templates on whichever computer does the compiling.

02 Build into a writable folder. For a successful local build, open **build/web** inside the generated project. For a remote build, extract the returned ZIP. Keep index.html, JavaScript, WebAssembly, game data and generated support files together.

03 To test locally on a desktop with Python, open a terminal in **build/web** and run **python3 -m http.server 8000 --bind 127.0.0.1**. Open **http://127.0.0.1:8000/** on that computer. This simple server is suitable for an ordinary local smoke test. Lua builds may need the isolation headers below.

04 For public hosting, serve the entire directory over HTTPS. The host should serve .wasm as application/wasm and JavaScript with a JavaScript MIME type. Test the hosted address, including the exact embedding page if applicable.

> PCA3D uses the Compatibility renderer and WebGL 2 for this target. For a game with Lua records, the export uses extension support and needs a secure, cross-origin-isolated context. Configure Cross-Origin-Opener-Policy: same-origin and Cross-Origin-Embedder-Policy: require-corp. Check the browser console for missing files or isolation errors.

REMOTE BUILD

Set up a private remote build worker

A worker is a computer you configure to accept a project, compile one requested target, and return the result. A phone’s Remote Build switch needs a working server at the other end.

01 Choose a trusted desktop for Android or Web, or a Mac for iOS. Configure Godot 4.7.2, templates, SDKs and signing there first. Make a local test build on that machine before introducing a network connection.

02 The distribution documentation describes a Python 3.10+ worker using **tools/build_worker/server.py** and **config.example.json**. In a distribution that includes them, copy the example to a private configuration file, set the full Godot path, and enable only targets the machine can build.

03 Set **PCA3D_BUILD_TOKEN** on the worker to a private random token of at least 32 characters. Start the documented server with **python3 tools/build_worker/server.py --config /path/to/worker.json**. Use the actual path on your system.

04 For a same-machine check, enter **http://127.0.0.1:8765** in Build Tools. From a phone or another computer, use the worker’s reachable HTTPS hostname with a trusted certificate. On a phone, 127.0.0.1 points back to the phone.

> The 0.86.3.4 Complete ZIP reviewed for this guide included the client and worker documentation, but the documented server script and example configuration were absent. If your package is the same, obtain the matching worker files from the distributor or connect to an existing compatible private worker. No hosted worker address is supplied by the switch.

REMOTE BUILD

Submit a job and collect its result

The client sends editable project material to the worker you selected. Choose a machine you trust with the game’s source and assets.

01 In Build Tools, enable **Remote Build**, paste the worker URL and current token, then set the target-specific options. The local Godot path is ignored for this job. Keep signing credentials on the worker.

02 Select the target in Build Game, choose the output parent, and keep the app active. A phone suspended by the OS can interrupt the transfer. The documented limit is 512 MiB for an upload, up to 1 GiB for the response and a 30-minute operation window.

03 Allow time for packaging, upload, compile and download. On success, read the result dialog’s archive path. The returned file ends in **-android-build.zip**, **-ios-build.zip** or **-web-build.zip**. Inspect the expected game.apk, game.ipa or index.html.

401 / authentication	Re-enter the worker’s current token. It is not persisted between editor sessions.
409 / busy	Wait for the current build. If your client timed out, check whether the worker is still compiling before submitting the same job again.
422 / compile failure	Read the worker’s error or export log. Check the SDK, templates and signing on the worker itself.
TLS / disconnected	Check the worker URL, certificate trust and network reachability from the client. An interrupted operation needs a new submission once the worker is ready.

BEFORE A MOBILE OR WEB EXPORT

Check media, scripts and rendering

A desktop preview can load media through a desktop-only extension. A mobile or browser target may need a different format or renderer before the same scene will work.

Movie and cutscene assets	In this build, the mobile/Web preparation step checks video assets and rejects non-.ogv video. Convert the media to Ogg Theora, update the actual asset reference, and replay its beginning, ending and audio.
Live2D / Cubism	The in-app mobile/Web path checks for recognized Cubism assets and blocks the export where compatible runtime support is missing. Simply removing a file from the finished build leaves the authored character broken.
Lua on the web	A project containing Lua records uses a Web template with extension support. An empty Lua collection uses the ordinary single-threaded path. The host and template both need to support the selected route.
Renderer-dependent effects	Check a representative scene using the target renderer. Browser output uses Compatibility. Pay particular attention to lighting, transparency, advanced fog and effects that looked different under desktop Forward+.
Input and storage	Test actual touch, keyboard or controller input on the target. Save, close and reopen the delivered game. In a browser, do this on the hosted origin players will use.

TROUBLESHOOTING

Read the first useful build error

Start by identifying which stage stopped: project validation, source preparation, Godot import, export, signing, transfer or launch. The right fix depends on that stage.

Missing project record	Open the validation report. Fix the specific ID or reference, apply, then rebuild. “Preview with errors” cannot make an invalid release build pass.
Missing or stale runner	Restore the runner bundle matching your PCA3D editor release. A similarly named executable from an older release may have the wrong runtime version or source digest.
Wrong Godot executable	In Build Tools choose the full editor required by your edition. For 0.86.3.4 the tool verifies Godot 4.7.2 stable. The executable from a finished game folder is a runner.
No export template found	Open export.log in the generated project. Install templates for the exact Godot version on the build computer. In the Web test here, the exporter named missing web_nothreads_debug.zip and web_nothreads_release.zip.
Import or compilation errors	Read import.log first if importing failed. If import succeeded, start with the first relevant error in export.log. Later engine messages may only describe the same failure.
macOS / Windows branding error	Read the named tool or missing dependency. On Linux the custom Windows icon step specifically reported that Wine was unavailable. The default-icon test isolated that step from the game itself.

VERIFICATION AND REFERENCE

Know when the build is ready to share

Keep the exported artifact, the target, and what you actually tested together. That makes “built,” “launched,” and “played on a device” three useful pieces of information.

Desktop packages	All three platform folders were generated using the default icon. The standalone Linux binary was launched and its authored room and HUD were observed. Windows and macOS execution were not tested.
Editable source	Standard, mobile and VR source outputs were created and inspected. Each had game/project.pca3d, project.godot and six export presets. The mobile touch setting and VR/OpenXR settings were checked.
Android, iOS and Web	The local iOS host check was observed on Linux. A local Web export reached Godot and stopped at missing templates. A missing worker token was detected by the client. Signed mobile apps, a finished Web build and a live remote worker were not verified here.

Before handing it to a player

Start from the delivered copy on the intended device. Reach the first playable room, interact, change rooms, save and reload. Include the whole platform folder or the full hosted Web output. Keep editable source or signing materials out of a player package unless the recipient specifically needs the source.

The steps and screenshots use PCA3D 0.86.3.4. The APK, IPA and Web in-app targets were introduced in 0.75.8. Older releases may offer source export where newer ones provide a direct target. Match templates and tools to your own editor release.

Official Godot export setup: Android · iOS · Web. Use the version selector for the Godot release installed on your build machine.