

Open-source find: UVTools makes the layers inspectable

A closer look at the project behind part of Seesaw's resin workflow, and a practical way to evaluate an existing tool before writing your own.



By Lyra

Published Oct 19, 2026 · 4 min read

October launch collection · Story context 2026-10-09



Sometimes the most useful GitHub find is a tool that exposes the part of a workflow you've been treating as a black box. For resin-print preparation, the layer file is a good place to look.

[UVTools is an open-source project](#) for inspecting, manipulating, converting, and checking resin-print files. Its documented checks include islands, resin traps, suction cups, and other layer issues.

Seesaw's October integration uses a pinned UVTools core alongside PrusaSlicer, giving us a concrete reason to explore the project.

This October review revisits an established tool through a recent integration. The upstream project shows the available tools; our integration shows how we put a small part of them to work.

| Start with the input it expects

Before adopting a library, identify the point where it enters your workflow. In Seesaw's resin path, PrusaSlicer prepares geometry and produces a sliced SL1 archive. UVTools then participates in inspection and printer-file encoding.

That boundary avoids a common integration mistake: assuming a tool covers every stage because its feature list is extensive. Read the accepted formats and inspect a minimal sample before designing the surrounding UI.

A useful evaluation note contains the input format, output format, version, invocation method, failure behavior, and license. Add one example file you are permitted to use. That note gives the next developer enough context to repeat the experiment.

| Look at the actual layers

A 3D viewport and a sliced layer viewer answer different questions. A layer inspection can reveal an isolated region or a cavity behavior that is hard to notice in the shaded model view.

Use a small fixture where you can predict what should appear. Step through neighboring layers and compare the issue report with the rendered data. If the tool flags a problem, make sure the wrapper preserves enough context for a person to understand it.

For developers, this is an observability lesson. Exposing the intermediate representation can shorten the distance between "the export failed" and "this region is disconnected at this stage."

| Treat repair as a policy decision

A tool offering repair features doesn't mean an application should enable every repair automatically. Decide which changes the product is allowed to make and how those changes will be checked.

Seesaw's documented Smart slice path uses a tightly bounded isolated-pixel repair rule and rejects structural islands. That is a product policy layered on top of the upstream capability. The wrapper needs to verify that the actual change stays within the product's repair policy.

Try the same approach in other domains. A document converter may offer aggressive cleanup; a data importer may infer missing values. Decide which transformations are acceptable for your users and preserve evidence of what changed.

| Try a small integration test

Create three fixtures: one valid, one with a targeted known issue, and one unsupported or malformed input. Run them through the exact integration path you plan to ship.

Check the exit result, the diagnostics, the output file, and any modified settings. Cancel a longer operation and verify that your application remains responsive and owns the cleanup. Then replace the output with a mismatched file and confirm your wrapper doesn't present it as the checked result.

These cases tell you more about your integration than a successful manual launch of the upstream desktop application. The product's responsibility begins where it invokes the tool and continues through the result it shows the user.

| Keep upstream visible

Pin the version used for the evidence, preserve the relevant license notices, and link the project. When reporting a bug, provide a minimal permitted fixture and enough environment detail for somebody else to reproduce it.

UVTools is worth exploring if you work with resin layer files. For us, the immediate value is being able to inspect the layers and understand a failed export. Keep physical test prints in the plan too; a checked file is one step on the way to a successful object.

Keep one accepted fixture unchanged between upgrades so each new engine version has a familiar comparison point.

| Sources & further reading

Checked 2026-10-14 · Story context 2026-10-09

1. [UVTools upstream project](#) (2026-10-14)
2. [Seesaw Smart slice policy](#) (2026-10-14)
3. [Seesaw backend validation](#) (2026-10-14)

[Read the current article online →](#)

Support the workshop

Explore software and learning offers at [StackSocial](#). Affiliate link: ProgreTech may earn a commission from purchases. Check current price, license and renewal terms before buying.

Find our projects at progretech.com.

ProgreTech LLC · Build / Learn / Improve / Together

This file contains its text, illustrations and fonts. No account, scripts or network connection required to read. External reference links need internet access.