# When the Lab Became a Module Builder

A build note on the moment Elliot Lab stopped exporting its test harness and started defining a real module boundary.

2026.04.01 00:31 DENNIS HEDEGREEN BUILD OPEN-NOTE
HTTPS://HEDEGREENRESEARCH.COM/ARTICLES/WHEN-THE-LAB-BECAME-A-MODULE-BUILDER/

The easiest false boundary in a new tool is often the one that looks obvious.

In Elliot Lab, it looked almost too obvious.

A `Source` on the left.
An `Output Basin` on the right.
What else would the module's public input and output be?

Inside the lab, that reading was useful.

At export time, it was wrong.

That is the architectural line Elliot Lab has now crossed.

The important change is not that the browser shell has more components, or that the water layer looks better, or even that Builder can now import more interesting scenes. The important change is that the tool stopped exporting its own test rig.

Before this correction, the shell was already doing real work. It could hold buckets, leaky buckets, siphons, reservoirs, and pipes. It could preserve conservation. It could show threshold, leak, blockage, return, and backpressure inside one closed scene. It could save and reopen that scene instead of throwing the whole state away after each run.

That was enough to say the project was no longer just a toy fluid sketch.

It was not yet enough to say the architecture was honest.

## The Harness Was Useful. It Was Still Wrong.

`Source` and `Output Basin` are not mistakes. They are good lab equipment.

They let me inject flow into a local scene.

They let me observe what leaves the local circuit.

They make a mechanism easy to press on and easy to read back.

That is exactly what a local harness should do.

The problem starts when that harness is treated as the thing being exported.

If the export carries the harness upward, the exported object is no longer only an internal mechanism with a public boundary. It is that mechanism plus the local scaffolding used to test it. The module is already contaminated by the bench it came from.

That is not just a naming problem.

It changes what the next layer is allowed to be.

Once the harness leaks into export, Builder stops behaving like a higher composition layer and starts behaving like a second screen full of half-disguised lab scenes. You are no longer importing modules. You are importing test setups that happen to contain modules somewhere inside them.

That was the real pressure point.

## The Boundary Moved Inward

The correction was simple in principle and serious in consequence.

Build Mode can now mark explicit public interface ports. `Auto Interface` can infer that boundary in simple scenes. And the export format now strips `Source`, `Output Basin`, and their local harness routes out of the exported module graph.

That changes the meaning of the whole stack.

Lab is where the internal mechanism is drawn, tuned, and stressed.

Export is where the public boundary is frozen.

Builder is where imported modules become black boxes with readable public ports instead of little lab diagrams pretending to be reusable units.

That is a more serious architecture because it is a more honest one. The tool is no longer saying that whatever was convenient for testing must also be part of the reusable object.

## The Multi-Input Case Made It Obvious

The clearest proof came from the small multi-input scenes.

When several harness `Source` components all point into the same internal bucket, the wrong question is: which one is the real module input?

That question still assumes the harness is the truth.

The better question is: how many public inputs should this module expose?

That is a different problem. It is an interface problem, not a bench-layout problem.

The first answer is intentionally narrow. Each harness `Source` can become its own public input. Each harness `Output Basin` can become its own public output. More than one public port may bind to the same internal port if that is how the actual scene is structured. Export v1 is capped at three inputs and three outputs.

That cap is not there because the idea is weak. It is there because the tool needs one honest interface rule before it earns the right to make larger claims.

This is where the project starts separating simulation semantics from interface semantics.

The water layer still matters. It is still the best way to make retention, threshold, leak, delay, and blockage visible. It is still the best way to feel when a scene is physically wrong or semantically confused.

But the public module boundary can no longer be defined by whatever merely looks like water input and water output on the screen.

## More Than A Feature Update

This is why the change matters more than a feature list would suggest.

Elliot Lab is still early.

Builder is still early.

The whole project is still much closer to a private workbench than to a finished design system.

But there is now a clean line through it:

- Lab for internal mechanism
- export for explicit public boundary
- Builder for composition above that boundary

That is enough to justify a stronger claim than "the simulator got better."

It became more serious when it learned what not to carry upward.

A lab becomes a module builder when it stops exporting its own test rig.

Elliot Lab on GitHub (https://github.com/DennisHedegreen/elliot-lab)

— Dennis Hedegreen

---

RELATION MEMORY

MATURES FROM The Making of Elliot · Moves from a single embodied-agent build note into a broader lab/module-builder pattern.

SYSTEM REQUIREMENT Elliot Was Never the Whole Program · Makes the lab/module layer explicit as the surrounding system Elliot required.

NEARBY A Tool Can Be Right and Wrong at the Same Time · Both articles treat tools as useful only when their limits stay visible.