What we changed, and why (plain-English summary)

This covers the work done to set up an automatic build & release process for Everest Forms Pro (and its ~40 addons) and Everest Forms (free), plus everything we found and fixed along the way.

The problem we started with

- **Pro had no automated release process at all.** To ship a new version of Pro or any addon, someone had to manually build it on their own computer, zip it, and upload it by hand. There was no automatic check that the version number matched, no record of what actually got built, and no way to know if a build was even broken until a customer complained.
- **Free had a release process, but it skipped a build step**, so a real release could go out without ever actually compiling fresh code — it would ship whatever happened to be sitting in the folder.
- **Nobody had a fast way to check "did this pull request break anything"** before merging it.

What we built

1. **A simple way to bump a version.** One command updates a product's version number and adds a "coming soon" changelog entry for you to fill in — instead of hand-editing multiple files and hoping you didn't typo the version number somewhere.

2. **An automatic draft release.** When someone finishes their changes and pushes a specific marker commit, a draft release gets created on GitHub by itself, with the changelog pulled in automatically from every product that has pending changes. A human still has to click "Publish" — nothing happens without a person approving it.

3. **An automatic build-and-ship process.** The moment that draft is published for real, every product mentioned in the changelog gets built fresh, zipped, and attached to the release automatically — instead of someone manually zipping 40 folders one at a time.

4. **A basic automatic check on every pull request**, for both Pro and Free: does the JavaScript follow the style rules, and does the code actually install cleanly on every PHP version we support (7.4 through 8.3)? This catches "works on my machine but breaks for some customers" problems before the code is even merged, instead of after a customer reports it.

5. **Simple "build everything" commands** so a developer can set up their local copy of the plugin with one command instead of remembering the exact sequence of steps for each of the ~40 products.

of the 40 products.

Real bugs we found and fixed along the way

None of these were things we broke — they were already sitting in the codebase, just never caught before because nothing was actually testing this thoroughly before.

- **A broken file that would have crashed the entire Pro plugin.** One core file had gotten corrupted at some point — a chunk of code was duplicated and mixed up inside itself. This is the kind of bug that causes a total white-screen crash the moment the plugin tries to load. It had been sitting there, undetected, because nothing had ever actually tried to build and load this file before. This is the single most important thing this whole effort caught.
- **A step that made every single build fail for no good reason.** A code- quality check was set up in a way that treated old, pre-existing style issues as a hard stop — so building almost any product would fail, even though nothing was actually wrong with it. Fixed so it reports problems without blocking the build.
- **One addon's download was 17x bigger than it needed to be** (32MB instead of 1.9MB) because it was bundling code for every single Google product (YouTube, Gmail, etc.) instead of just the one feature (Google Sheets) it actually uses. Trimmed it down to just what's needed.
- **A step that ran for every product even when it wasn't needed**, wasting time and occasionally failing builds that had nothing to do with it. Fixed so it only runs for the two products that actually need it.
- **A release's changelog could point at the wrong version of the code** if it got updated more than once before being published — fixed so it always stays pointed at the latest version.
- **Uploading a file to a release could fail** because the automation didn't tell it clearly enough which repository to upload to. Fixed.
- **Compiled files (CSS and JavaScript) were being saved directly into the project's history every time**, even though they're just the "cooked" output of source files that are already saved — like re-saving a photo's thumbnail every time you touch the original photo. This made every build look like it had "changed" hundreds of files, when really nothing meaningful had changed. We stopped saving those cooked files and left the real source files in place — the cooked versions get regenerated automatically whenever a real build runs, so nothing is lost.
    - The one thing this changes for developers: after downloading a fresh copy of the code, you now need to run the build command once before testing in a browser, otherwise it'll look unstyled. Before, it would look "ready" immediately because those cooked files were already sitting there — but that also meant they were quietly going stale and

were already sitting there — but that also meant they were quietly going stale and being re-saved unnecessarily.

What we deliberately did NOT do

- We never published a single real release, published anything to WordPress.org, or changed anything customers can see, while testing any of this. Every test used disposable, clearly-labeled test branches and releases that were deleted afterward.
- We left a couple of small, known, low-risk items alone on purpose (like one addon's dependency being bigger than ideal, but not worth the risk of a fix that could break it) rather than rushing a fix under time pressure.

Why this matters

Before: releasing anything was manual, error-prone, and nobody would know something was broken until a customer found it. Now: every release is built and checked the same way, every time, automatically — and the very first time this got tested properly, it caught a bug serious enough to crash the entire plugin. That's the actual payoff.