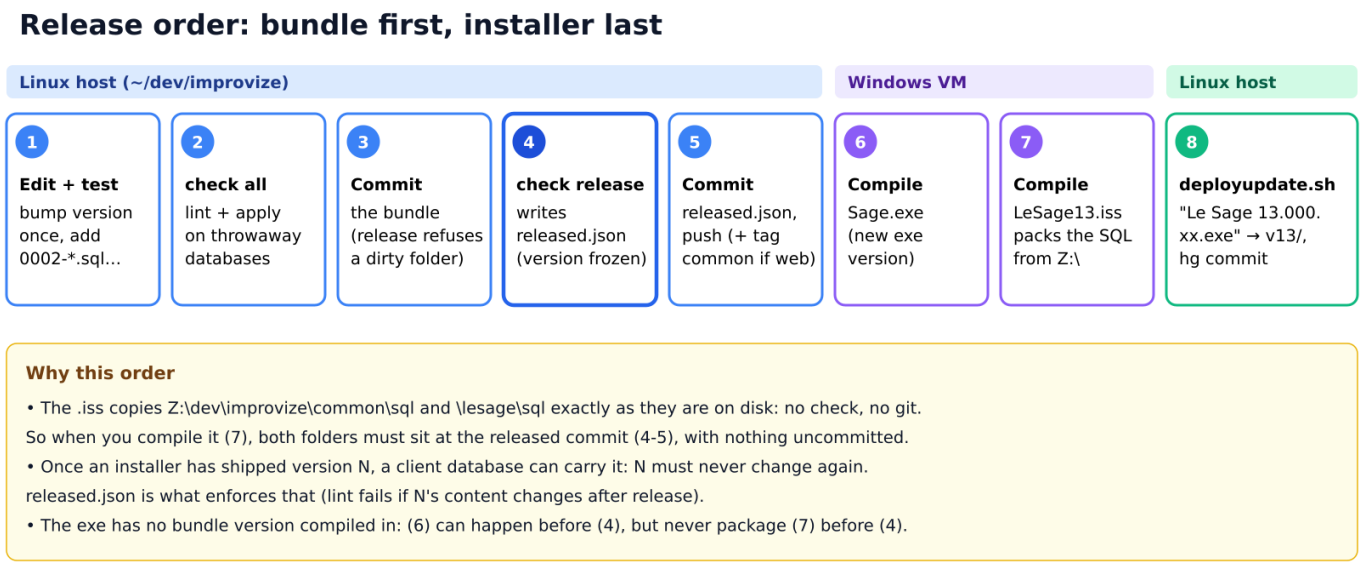
Releasing a bundle and building the installer

"Releasing" a bundle version means freezing it: recording its checksum and git commit in `released.json`, because from then on a real database may carry it. Delphi installers ship **only released versions**.



Checklist

On the Linux host

1. **Version bumped**: `lesage/sql/version` is above the last entry of `lesage/released.json` (same for `common` if you changed it). New steps are named `NNNN-*.sql` with that number.
2. **Everything passes**:

   ```
   cd ~/dev/improvize/common
   export PGHOST=/var/run/postgresql
   npm run check -- all
   ```

3. **Commit the bundle** (and `functions.list`, `data/`) and push. `check release` refuses a bundle folder with uncommitted changes, and records the current `HEAD`.

```
cd ~/dev/improvize/lesage
git add sql data-src && git commit -m "feat(sql): …" && git push
```

4. **Release**: runs lint and apply again, then writes the new entry in each `released.json` whose version is new (an already released version is just reported).

   ```
   cd ~/dev/improvize/common
   npm run check -- release sql ../lesage/sql
   ```

   Last line expected: `lesage: released version N sha256:… at <commit> — commit released.json`.
5. **Commit** `released.json` in each repo it touched, and push:

   ```
   cd ~/dev/improvize/lesage
   git add released.json && git commit -m "chore(release): lesage bundle version 2" &&
   git push
   ```

   The lesage-web commit hook runs prettier on `released.json` and reflows it: harmless, the lint compares the values only (`node scripts/lint-bundles.mjs` still says `ok`). If `common` was released too: same in `~/dev/improvize/common`, and tag it (`v1.x.y`) when the web apps must pick it up (they pin `improvize-db-bundles#semver:^1.0.0`, then `npm update improvize-db-bundles`).

In the Windows VM

6. **Compile** `Sage.exe` with its new exe version (`versionprefix.txt` → `genversion.cmd`, which writes `version.RES` and the `version.ini` the `.iss` reads).
7. **Compile** `LeSage13.iss`. It copies `Z:\dev\improvize\common\sql\*` → `{app}\common\sql` and `Z:\dev\improvize\lesage\sql\*` → `{app}\lesage\sql`, after wiping both folders (`[InstallDelete]`) so no stale file survives. Output: `K:\Dev\Delphi\Sage13\Le Sage 13.000.xx.exe`.

Back on the host

8. **Publish**: `./deployupdate.sh "Le Sage 13.000.xx.exe"` in `Sage13/` (Mercurial commit + push, upload to `…/lesage/program/v13/`, `makelatest.pl`, announcement mail).

Why the bundle comes first

- The `.iss` does no check at all: it packs whatever is in the two folders at compile time. If you compile it while you have uncommitted or unreleased SQL there, that SQL goes to clients and becomes a version nobody can reproduce.

- Before step 7: `git -C ~/dev/improvize/lesage status sql` and `git -C ~/dev/improvize/common status sql` must be clean.
- The exe has no bundle version compiled in. Compiling `Sage.exe` before the release is fine; **packaging** before it is not.

Common situations

Situation	Do
Exe-only fix, no SQL change	No release. Bundles are already at a released commit; just steps 6–8.
Bug in a released version N	Never edit N. Bump to N+1 and fix it with a new step or a corrected function.
`check release` says "uncommitted changes in the bundle folder"	Commit (or stash) first: it records the commit, so the folder must match it.
Lint: `released` rule fails	You changed a released version's content without bumping `version`.
Changed only `common`	Release `common`, then rebuild the installer of **every** product (Le Sage, Inpro, Aïda): each ships its own copy.

> Never put a 13 exe in `program/v12/`: `makelatest.pl` would offer it to every 12.x, ready database or not. 13 goes in `v13/` only.

Revision #3
Created 2026-11-18 14:55:38 UTC by Claude-Aida
Updated 2026-11-18 15:43:07 UTC by Claude-Aida