Private modules and the `-internals` convention

- What -internals means
- Where it goes
- Choosing
- Splitting a shared mechanism out

What `-internals` means

A folder named `-internals` holds code that is **not public API**. It may be renamed, reshaped or deleted without notice or a deprecation cycle. Plugins and themes must never depend on it.

Only the folder's owner imports from it. Everything else, including core code that uses a primitive, imports the owner's public entry instead. The UI kit guide states the same rule for the kit's primitives. Nothing lints this, so the folder name is the only signal.

Where it goes

Put `-internals` immediately inside the unit that **owns the concept**. Ownership decides placement, not who imports it.

One named owner. A component or module owns the concept, so its internals sit inside it, and only that owner imports them:

```
ui-kit/d-reorderable-list/-internals/     owned by DReorderableList
lib/blocks/-internals/                    owned by the blocks module
```

The UI kit guide's "Splitting a large primitive" section describes how a primitive lays out its own `-internals` folder.

No single owner. Several members share the mechanism, so it goes in the `-internals` of the smallest namespace that contains every consumer, under a folder named for the mechanism. The owner is then that namespace, and any member of it may import the folder:

```
ui-kit/-internals/cursor/         shared by dRovingFocus and DReorderableList,
                                  both ui-kit members
lib/-internals/drag-and-drop/     shared by ui-kit modifiers and a service, so no
                                  namespace narrower than the app contains them
```

`lib/` is the app-wide root, which is why a mechanism whose consumers span several namespaces lands there. Reach for it only when nothing narrower fits. A mechanism used only within `ui-kit` belongs to `ui-kit`.

Choosing

1. Does one component or module own the concept? Then `<owner>/-internals/`.
2. Otherwise, list every consumer and take the smallest namespace containing all of them. Then `<namespace>/-internals/<mechanism>/`.
3. If that namespace is the app itself, it is `lib/-internals/<mechanism>/`.

Do not nest `-internals` under a namespace-scoped `lib/`. A namespace `lib/` (as in `form-kit/lib/`) holds flat, supported helpers. `-internals` already says the code is private, so the extra level adds nothing.

Splitting a shared mechanism out

When a second consumer appears for something inside one owner's `-internals`, move the shared part up rather than importing across. Keep the move behaviour-neutral and let the existing suite prove it. `ItemScope` and the stepping helpers were lifted out of `ui-kit/modifiers/d-roving-focus/` into `ui-kit/-internals/cursor/` this way, with the modifier's own tests as the gate.

A shared mechanism folder keeps its types next to the code that uses them. Add a `types.ts` only when several files in the folder read from it.

□□□ #2
□ □□□ □□□ 2026-10-31 13:28:50 UTC
□ □□□ □□□ 2026-10-31 13:28:51 UTC