Release Strategy

Unified versioning and release management

Current: Main Branch

- Developer merges PR to main with conventional commits
- release-please analyzes commits since last release
- Determines version bump: `feat:` → minor, `fix:` → patch, `BREAKING CHANGE:` → major
- Opens/updates release PR (updates CHANGELOG.md and package.json)
- Merge creates prerelease tagged `v1.25.0-beta`
- Triggers `release/X.Y.x` branch creation
- Empty commit with `Release-As: X.Y.0` removes `-beta` suffix

Current: Hotfix Workflow

- Push to `release/**` branch
- release-please auto-bumps patch
- Creates release PR
- Merge creates stable release

Current Challenges

Independent versioning per workspace

Package.json: 1.25.0-beta

Controls apps: 0.1.0

Firmware: 0.1.0

Explorer: 1.0.0

No single version number represents the entire system state

Benefits Worth Preserving

Automated release process

Conventional commit based

Changelog generation included

Separate workflows for main and release branches

New Strategy: Unified Versioning

All version strings synced:

- package.json at repo root
- controls workspace Cargo.toml
- firmware workspace Cargo.toml
- explorer workspace Cargo.toml
- cabinet-controller Cargo.toml

Release Process

When we want to make a release:

- Bump all version strings uniformly, update changelog, commit
- Create release branch - tags commit and creates branch (e.g., `release/1.26.x`)
- Publish to GitHub - creates GitHub release

Hotfix Process

To apply hotfixes to release branches:

- Cherry-pick the SHA of the commit and bump the patch
- Publish that release to GitHub (e.g., `v1.26.1`)

The makeline-release Tool

I made a tool called makeline-release that does this reliably

Just commands wrap each step

This entirely replaces release-please

Commands: Version Bumps

```
# Update version in all 5 places
# And update CHANGELOG.md at repo root
just bump-minor-version          # 1.25.0 → 1.26.0
just bump-major-version          # 1.25.0 → 2.0.0
```

CHANGELOG.md automatically updated during bump step

Commands: Create Release Branch

```
# Tag current commit and create release branch
just create-release-branch        # Tags and creates release/1.26.x

# Or with suffix
just create-release-branch beta   # release/1.26.x-beta
```

Commands: Publish Release

```
# Create GitHub release from existing tag
just publish-release                    # Creates GitHub release v1.26.0
```

Development continues on main

Commands: Hotfixes

```
# Find commits to backport
git log main --oneline

# Apply hotfix - bumps patch version
just hotfix <sha>

# Publish release with hotfixes
just publish-release                # v1.26.1
```

Commands: Dry Run Mode

You can append -dry to any command:

```
just bump-minor-version-dry
just bump-major-version-dry
just create-release-branch-dry
just publish-release-dry
```

CI Integration

Publishing with `release/**` branch name triggers existing workflows:

- Firmware artifact build and upload
- Greengrass component compilation
- Greengrass component deployment
- Artifact uploading to GitHub release

Everything continues to work!

What We Get

- Uniform semantic versioning
- Tags on main branch commits
- Release branches made from those tagged commits
- Ability to add hotfixes to release branches and publish new releases including them

Key Differences

Versioning: Independent per workspace → Single unified version

Release coupling: Release PRs must merge to main → Releases decoupled from main

Control: Automatic release-please → Explicit version bumps

Changelog: release-please → git-cliff

Both: Conventional commits required

Benefits

Preserves: Automated changelogs, conventional commits, main/release workflows

Adds:

- Unified versioning across all components
- Development decoupled from releases (no release PRs on main)
- Explicit control over when to release
- Simpler hotfixes
- Tags on main mark specific commits as releases

Questions?