Release Strategy

Unified versioning and release management

Current Release Strategy

Uses release-please for automated releases

Triggered on every push to main

Creates release PRs automatically

Current Versioning

Version managed in package.json

Controls, firmware, and explorer have independent versions

Main branch releases use `-beta` suffix

Example: `1.25.0-beta`

Current Release Process

1. Developers merge PRs to main
2. release-please analyzes commits
3. Opens/updates a release PR
4. Merge release PR to create release
5. Release creation triggers branch workflow
6. Creates `release/X.Y.x` branch

Current Branch Creation

When a prerelease is published:

- Extracts major.minor from version
- Creates branch `release/X.Y.x`
- Commits with `Release-As: X.Y.0`
- Bumps to stable version on branch

Current Changelog

Automatically generated from conventional commits

Groups by commit type (feat, fix, etc.)

Links to commits and Pull Requests

Updated with each release PR

Current Hotfix Workflow

Automated via release-please

Runs on push to release/** branches

Always bumps patch version on release branches

Creates release PRs on release branches

Current Release Branch Behavior

Separate release-please configuration

Uses always-bump-patch versioning

No manual version control needed

Relies on conventional commits

Considerations

Independent versioning per workspace

Package.json: 1.25.0-beta

Controls: workspace shared version

Firmware: 0.1.0

Explorer: 1.0.0

Current Approach Benefits

Automated release process

Conventional commit based

Changelog generation included

Separate workflows for main and release branches

What Changed?

Version strings are now synced across:

- Package.json (repo root)
- Controls workspace Cargo.toml
- Firmware workspace Cargo.toml
- Explorer workspace Cargo.toml
- Cabinet-controller workspace Cargo.toml

Release Process

1. Bump all version strings uniformly
2. Update the changelog
3. Commit changes
4. Tag that commit (e.g., `v1.2.0`)
5. Create release branch (e.g., `release/hyphenx-v1.2.x`)
6. Publish release to GitHub

Hotfix Process

To apply hotfixes to release branches:

1. Cherry-pick the SHA of the commit
2. Bump the patch version
3. Publish release to GitHub

The makeline-release Tool

Custom tool that handles releases reliably

Just commands wrap each step for convenience

Bump Version

```
# Update version in all 5 places
# And update CHANGELOG.md

just bump-minor-version  # 1.25.0 → 1.26.0
just bump-major-version  # 1.25.0 → 2.0.0
```

Create Release Branch

```
# Tag current commit on main
# Create/switch to release branch

just create-release-branch        # release/1.26.x

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

Publish Release

```
# Publish the release to GitHub

just publish-release  # Creates hyphenx-v1.26.x
```

Apply Hotfixes

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

# Apply hotfix (bumps patch version)
just hotfix <sha>

# Publish updated release
just publish-release
```

Dry Run Mode

All commands support dry run:

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

Changelog Management

CHANGELOG.md is automatically updated during version bump

Uses git-cliff for changelog generation

No manual changelog editing required

CI Integration

Publishing with `release/**` branch triggers:

- Existing CI workflows
- Artifact uploading
- Greengrass component deployment

Everything continues to work!

Benefits

- Uniform semantic versioning
- Tags on main branch commits
- Release branches from tagged commits
- Hotfix capability
- Automated changelog management

Replaces release-please

This entirely replaces the old release-please workflow

Why This Approach?

- Unified version for software suite
- Better control over release process
- Simpler hotfix management
- Clearer release history

Questions?