Versioning

Introduction

Semantic Versioning (SemVer) is a versioning specification that defines how to increment version numbers based on the types of changes. SemVer, when strictly adhered to, attempts to put the burden of determining if a change in a particular package will break consuming packages on the maintainer of the original package.

This allows consuming packages to understand if a new version contains breaking changes (a major version increment), backwards-compatible functional changes (a minor version increment), or backwards compatible bug fixes (patch version increment).

Authors who adhere to semver conventions allow consumers to integrate patch and minor versions changes with little or no risk. It also notifies consumers when a new version could require changes to consumers. This helps alleviate update phobia and encourages consumers to update quickly. In turn, this promotes rapid bug and security fixes.

INFO

All UDS packages adhere to the SemVer versioning scheme.

Beachball

UDS uses beachball to manage versioning and publishing of packages to npm.

INFO

A beachball change file should be included in all UDS PRs. See the repo readme for how to add a change file.

Beachball uses change files to allow developers to indicate their intent when making a change. The intent includes:

- a description of the change that will be included in the changelog
- a semver bump (`major`, `minor`, `patch`) that the change represents

This decouples git commit messages from the versioning and release process as is common with conventional commits and standard-version. By decoupling versioning and releasing from git, developers can include as much info as they require in their git commit messages and use whatever git workflows are desired (allowing squashing and modifying commits to be completely safe).

By using change files, developers can include whatever and as much important information as is required about a change because the change message will be added directly to the changelog. This also removes the abstraction that maps conventional commit prefixes (`fix`, `feature`, etc) to semver bumps and requires describing the change directly in terms of semver (`major`, `minor`, `patch`). Finally, since change files are stored on the "disk", they can be reviewed and modified easily right up to the time of a release 🪄 .

See the Releasing ADR in the repo for more information about why beachball was chosen to manage versioning and publishing of npm packages.

> INFO
>
> Descriptive commit messages, especially when using squash and merge to merge PRs in GitHub, are are still important to ensure a useful git history. Developers are entrusted to be well intentioned 😄 .

Palette and theme SemVer increments

While by default, most UDS packages use conventional commits to calculate SemVer increments during the release cycle, palette and theme packages implement some custom logic to calculate their version increments.

The general rule for UDS is that any visual change, no matter how important, should be tested manually by consumers and is considered a major version increment. For palettes and themes, removing or modifying any design tokens can result in a visual change for consumers. Using this mentality, the following table describes the different types of SemVer increments for palettes and themes:

Type of change	SemVer increment
Modifying a design token	major
Removing a design token	major
Adding a design token	minor
Adding, removing, or modifying documentation (descriptions)	patch

Theme schema versioning

Themes and UDS base components are tightly coupled. Components expect themes to provide a defined set of design tokens. The `system-theme-tokens` package abstracts this coupling into a separate package so that themes and components can have independent release cycles.

The `system-theme-tokens` package defines the tokens required for each component. In this way it versions the schema that links the `components-base` package to the various themes. The various themes and the `components-base` both have a dependency on `system-theme-tokens`. When a new theme version is published, it uses a specific version of the `system-theme-tokens` package to build and validate it's design tokens against. The version of `system-theme-tokens` used is stored in the metadata of the built theme. When the `base-components` load a theme using the ThemeProvider, the version of `system-theme-tokens` stored in the theme metadata is validated to be semver compatible with the version of `system-theme-tokens` that the `components-base` depends on. The semver.satisfies function is used to ensure compatibility between the versions.

Refer to https://github.com/npm/node-semver and https://semver.npmjs.com/ for details about how semver compatibility works.

⚠ CAUTION

If a theme is loaded that was not built with a compatible version of `system-theme-tokens` the `base-components` will throw an error and the components will not be rendered.

This is intentional because a semver incompatible theme will have a mismatch on the design tokens (too few or too many) that components expect. Therefore, the `base-components` fail fast rather than allowing for the potential of errors to occur due to design token mismatching that could be potentially difficult to debug.

Edit this page