

# Semantic Versioning



Mike DiDomizio, June 2024

# What is Semantic Versioning (SemVer)?

- A versioning system that follows the format MAJOR.MINOR.PATCH ex. 2.3.3.
- Commonly used by packages in dependency management systems like NPM, Maven, Homebrew
- Visually details the type of change between versions.
  - **MAJOR**: Breaking changes - 1.2.0 => 2.0.0
  - **MINOR**: New features, backward-compatible => 1.0.0 => 1.1.0
  - **PATCH**: Bug fixes, backward-compatible => 1.0.0 => 1.0.1

# Why is versioning important?

- Tracking changes over time
- Managing dependencies
- Communicating changes
- Enables rollbacks
- Provides legal and compliance tracking

# Major changes


- Generally existing APIs have been changed from how they previously worked. For example, instead of accepting a string value, it now expects a number
- Means that upgrading the version could mean updates are necessary in your code
- Examples:
  - `POST /api/sport { "sport": 2 }` => `POST /api/sport { "sport": "hockey" }`
  - UI components `
- Can include new features, bug fixes
- Resets the **MINOR** and **PATCH** version - 4.6.3 => 5.0.0



```
- function getSport(sport: number) {  
+ function getSport(sport: string) {  
}
```

# Minor changes

- Generally means a new feature or functionality that is backwards compatible
- Examples:
  - Adding a new API endpoint  
`GET /api/sports`
  - Creating a new UI component
  - Adding new functionality to an existing component
- Resets the **PATCH** version - 4.8.3 => 4.9.0



```
- function getSport(sport) {  
+ function getSport(sport, format) {  
+   if (format === 'json') {  
+     return formattedData  
+   }  
  
  return nonFormattedData  
}
```

# Patch changes

- Adds a fix/change to existing functionality. Fixing logic that throws errors when it was not supposed to.
- Increments to next number - 4.6.2 => 4.6.3
- Patch changes or bug fixes, can be applied to multiple different versions, but it is not a requirement.  
ex. 4.6.2 => 4.6.3  
4.5.8 => 4.5.9



```
function getSport(sport) {  
- if (sport === 'hcokey') {  
+ if (sport === 'hockey') {  
  }  
}
```

# Prerelease versioning and build metadata

- Used to denote `alpha`, `beta`, `canary`, `release candidate` builds  
versions  
major.minor.patch+buildMetadata (e.g., 1.0.0+202406091000).  
major.minor.patch-rc.1
- Prerelease versioning indicates the progression of development and testing before a final release, affecting the version's position in the release cycle.
- Use build metadata for identifying specific builds, without affecting version precedence. Useful for internal tracking or CI/CD processes.
- Not a requirement to use

# Best Practices

- Take the versioning out of people's hands with tools like ChangeSets.
  - Less chance of making mistakes - 2.3.4 => 2.3.6 oops forgot 2.3.5
- Shouldn't unpublish packages or versions, or overwrite versions
  - Ensures consistency
  - Unpublishing can cause rippling effects - NPM history of left-pad. With NPM this can no longer be done (for most circumstances).
  - Can deprecate a version that you no longer want people to use.
    - In NPM, this will output a message on install that a package is deprecated
- CHANGELOG can be helpful to tell users what a release contains



# Common pitfalls

- Inconsistent versioning practices  
Example: Devs not versioning things properly - minor for bug fixes, patch for new functionality. It can lead to confusion on what the new version contains. Can be dangerous to users, cause confusion or distrust.
- Misuse of **MAJOR** version changes. Hard to understand how drastically something has changed between versions. Can cause hesitancy for users to upgrade, lack of new version adoption due to always being a major version change.
- Lots of **MAJOR** version changes - Can cause exhaustion for users keeping up with changes, distrust, be a sign of poor planning. A user may choose a package that has consistency and stability over yours.
- Lack of information in CHANGELOG - Can cause hesitancy of adoption if the user doesn't know what is changing.

# Common misconceptions

- Anything less than version 1 is unusable ex. 0.1.0
  - Generally used to tell users “This is probably unstable, and may be changing”, initial alpha/beta version.
  - Common to start with 1.0.0 as it is the “first version”, but there is no hard rule.
- All changes require a version bump - you simply don’t have to publish on every change. You can choose to group changes into one release.
- **PATCH** changes are only for fixing bugs - can be for publishing improvements to documentation, or refactoring/optimizations.
- Numbers cannot go above 10 - When you get to 2.4.9, the next change is not necessarily 2.5.0. A fix version bump would be 2.4.10.
- A **MINOR** or **PATCH** version guarantees backwards compatibility. Versioning is still handled by humans, and people make mistakes. Sometimes bug fixes introduce other bugs that were not intended.

# Common alternatives to Semantic Versioning

- CalVer - uses the calendar date as the version number. Easy to understand when a version was released.
  - Example: 2024.06.1 (June 1, 2024), 23.1.15 (January 15, 2023).
- MAJOR.MINOR
  - Example: 2.5 (Major version 2, Minor version 5).
  - Less granular than SemVer
- Rolling version - Uses a continuous, incrementing number, often without major/minor differentiation.
  - Example: build-1234, revision-5678.

# Other uses for versioning

- Documentation and publications (notes, documents, books)
- Public APIs (GitHub)
- Business processes and policies
- Hardware
- Online Courses

You can version anything really!

# MAJOR.MINOR.PATCH recap

- **MAJOR** - Something has changed that is no longer backwards compatible, upgrading may break my product.
- **MINOR** - A new separate functionality has been added. Should not be a problem to upgrade.
- **PATCH** - This is a fix so no new functionality has been added, but something has been changed. Should not be a problem to upgrade.

# Versioning gone mad

## Because why not

- Xbox, Xbox 360, Xbox One, Xbox One S, Xbox One X, Xbox Series X, Xbox Series S.
- Windows 3.1, Windows 95, Windows 98, Windows XP, Windows 2000, Windows ME, Windows Vista, Windows 7, Windows 8, Windows 8.1, Windows 10
- Battlefield 1942, Battlefield 2, Battlefield 1943, Battlefield 3, Battlefield 4, Battlefield 2142, Battlefield 1, Battlefield V, Battlefield 2042
- Modern Warfare 2 (2009), Modern Warfare 2 (2020)

Having inconsistent versioning can make things hard for people to understand.

# Conclusion

Versioning software is a fundamental practice in software development that ensures clarity, reliability, and efficiency in managing software lifecycle processes. It provides a structured way to handle updates, communicate changes, and maintain compatibility, significantly contributing to the overall success of software projects.

# Thanks for reading this!

Let me know if you have any feedback

- Excellent and maybe the only resource needed for detailed information on Semantic Versioning.
- There is a 12 question quiz 🙄. It should take a few minutes to go through and it is anonymous. Validate what you've learned!



Mike DiDomizio, June 2024