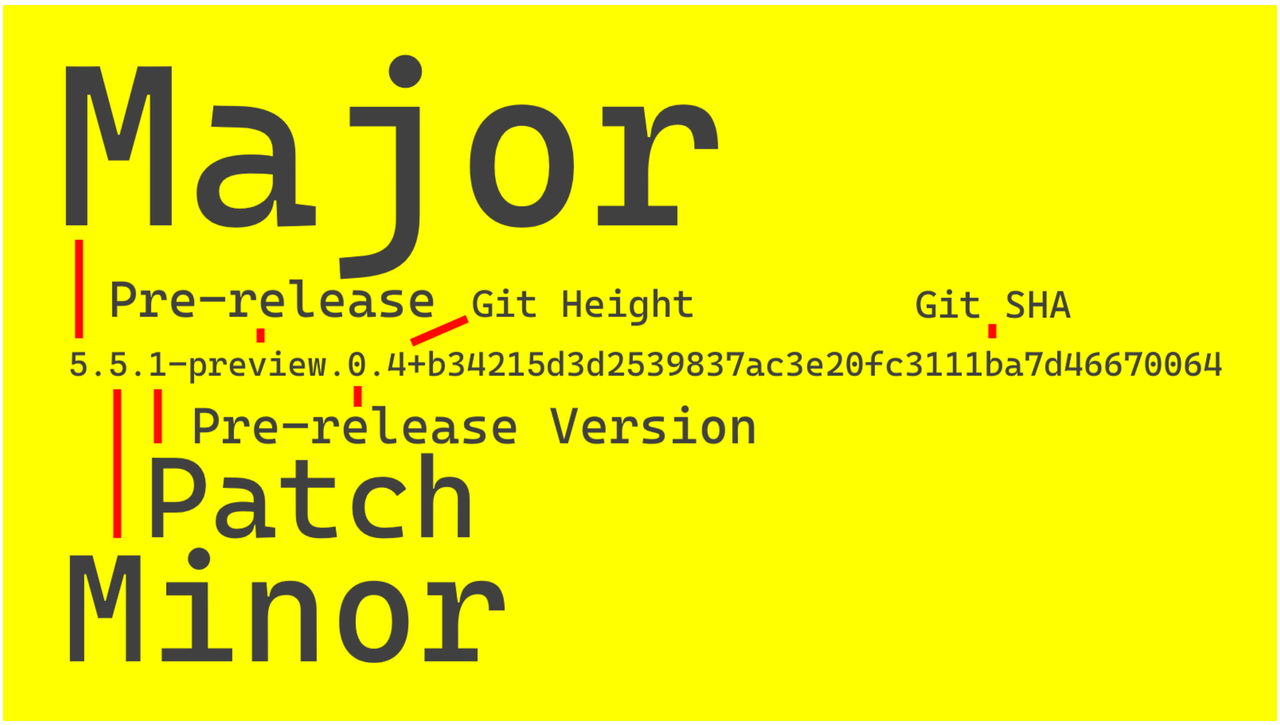
The Easiest Way to Version NuGet Packages

Skip to content

Posted 1 July - **5 min read**



The **easiest** way to version NuGet packages using semantic versioning in my opinion is to use MinVer. Getting started is literally as easy as adding the `MinVer` NuGet package. Getting finished is not too much more than that.

In this post I'll discuss the semantic versioning 2.0 standard and show you how you can semantically version your NuGet packages and the DLL's within them MinVer and Git tags.

What is Semantic Versioning?

Semantic versioning is the idea that each part of a version number has some intrinsic meaning. Lets break down an example of a full version number into it's constituent parts:

```
1.2.3-preview.0.4+b34215d3d2539837ac3e20fc3111ba7d46670064
```

- **1** - The major version number. Incrementing this means that a major breaking change has occurred.
- **2** - The minor version number. Incrementing this means that a non-breaking change has occurred.

- **3** - The patch version number. Incrementing this means that a patch or fix has been issued for a bug.
- **preview** (Optional) - This determines that the build is a pre-release build. This pre-release label is often set to `alpha` or `beta`.
- **0** (Optional) - This is the pre-release version number.
- **4** (Optional) - The Git height or the number of commits since the last non-pre-release build.
- **b34215d3d2539837ac3e20fc3111ba7d46670064** (Optional) - The Git SHA or hash of the current commit.

Isn't that cool! Every number in there has so much significance. Lets break it down, just by looking at version numbers we can determine:

- Whether something was fixed, enhanced or broken when comparing one version to the previous one.
- Whether it is a release or pre-release version.
- Which commit the code was built with.
- How many commits were made after the last release.

Versioning the Wrong Way

In the past I've tried to generate version numbers in quite a few different ways, none of which has been very satisfactory and none have conformed to semantic versioning 2.0. I've tried using the current date and time to generate a version number. This tells you when the package was created but nothing more.

```
[Year].[Month].[Day].[Hour][Minutes]
2020.7.2.0908
```

I've also generated version numbers based on the automatically incrementing continuous integration (CI) build number but how do you turn one number into three? Well in my case I hard coded a major or minor version and used the CI build number for the patch version. Using this method lets you tie a package version back to a CI build and through inference a Git commit but it's less than ideal.

```
[Hard Coded].[Hard Coded].[CI Build Number]
1.2.3
```

MinVer

MinVer leans on Git tags to help version your NuGet packages and the assemblies within them. Lets start by adding the MinVer NuGet package to a new class library project:

```
<ItemGroup Label="Package References">
  <PackageReference Include="MinVer" PrivateAssets="All" Version="2.3.0" />
</ItemGroup>
```

We'll need an initial version number for our NuGet package, so I'll tag the current commit as `0.0.1` and push the tag to my Git repository. Then I'll build my NuGet package:

```
git tag -a 0.0.1 -m "Initial"
git push --tags
dotnet build
```

If you now use an IL decompiler tool like dnSpy (which is free and open source) to take a peek inside the resulting DLL, you'll notice the following version assembly level attributes have been automatically added:

```
[assembly: AssemblyVersion("0.0.0.0")]
[assembly: AssemblyFileVersion("0.0.1.0")]
[assembly:
AssemblyInformationalVersion("0.0.1+362b09133bfbad28ef8a015c634efdb35eb17122")]
```

If you now run `dotnet pack` to build a NuGet package, you'll notice that it has the correct version. Note that `0.0.1` is a **release** version of our NuGet package i.e. something we might want push to nuget.org in this case.

Now lets make a random change in our repository and then rebuild and repack our NuGet package:

```
git add .
git commit -m "Some changes"
dotnet build
dotnet pack
```

Now MinVer has automatically generated a **pre-release** version of our NuGet package. The patch version has been automatically incremented, a pre-release name `preview` has been given with a pre-release version of `0`. We also have a git height of one because we have made one commit since our last release and we still have the git commit SHA too:

If we crack open our DLL and view it's assembly level attributes again, we'll see more details:

```
[assembly: AssemblyVersion("0.0.0.0")]
[assembly: AssemblyFileVersion("0.0.2.0")]
[assembly: AssemblyInformationalVersion("0.0.2-
preview.0.1+7af23ee0f769ddf0eb8991d59ad09dcbc8d82855")]
```

Now at this stage you could make some more commits and you'd see the major, minor and patch versions stay the same but the preview version, git height and git SHA would change. Eventually though, you will want to get another **release** version of your NuGet package ready. Well, this is as simple as creating another git tag:

```
git tag -a 0.0.2 -m "The next amazing version"
git push --tags
dotnet build
```

Now you can simply take the latest `0.0.2` release and push it to nuget.org.

Nerdbank.GitVersioning

There is a more popular competitor to MinVer out there called Nerdbank.GitVersioning which is part of the .NET Foundation and is worth talking about because it works slightly differently. It requires you to have a `version.json` file in your repository to contain the version information, instead of using Git tags.

```
{
  "$schema":
"https://raw.githubusercontent.com/dotnet/Nerdbank.GitVersioning/master/src/NerdBank.GitVe

  "version": "1.0-beta"
  // This file can get very complicated...
}
```

In my opinion, this is not as nice. Git tags are an underused feature of Git and using them to tag release versions of your packages is a great use case. Git allows you to checkout code from a tag, so you can easily view the code in a package just by knowing it's version.

```
git checkout 0.0.1
```

Having a version number in a file, also means lots of commits just to edit the version number.

Conclusions

MinVer is an easy way to version you NuGet packages and DLL's. It also comes with a CLI tool that you can use to version other things like Docker images which I'll cover in another post. If you'd like to see an example of MinVer in action, you can try my Dotnet Boxed NuGet package project template by running a few simple commands to create a new project:

```
dotnet new --install Boxed.Templates
dotnet new nuget --name "MyProject"
```

Web Mentions

What's this?
0 Likes

0 Reposts

0 Linked

0 Replies

Comment

No comments yet. Leave the first comment !

Newsletter

Muhammad Rehan Saeed

Software Developer at Microsoft, Open Source Contributor and Blogger