

## Ruby master - Feature #15456

### Adopt some kind of consistent versioning mechanism

12/23/2018 09:29 PM - ioquatix (Samuel Williams)

|   |        |
|---|--------|
| <b>Status:</b>  | Open   |
| <b>Priority:</b>  | Normal |
| <b>Assignee:</b>  |        |
| <b>Target version:</b>  |        |
| <b>Description</b>  |        |
| <p>After the discussion <a href="https://github.com/ruby/bigdecimal/issues/114">https://github.com/ruby/bigdecimal/issues/114</a> I feel like we would benefit from some consistent versioning mechanism across all of Ruby.</p> <p>So far, I feel the majority of Ruby uses some form of semantic versioning.</p> <p>For the sanity of all Ruby users, I think it would be a good policy to adopt this across core Ruby and standard gems.</p> <p>There are some previous discussions around this:</p> <ul style="list-style-type: none"><li>• <a href="https://bugs.ruby-lang.org/issues/9215">https://bugs.ruby-lang.org/issues/9215</a></li><li>• <a href="https://bugs.ruby-lang.org/projects/ruby/wiki/GeneralMaintenancePolicy">https://bugs.ruby-lang.org/projects/ruby/wiki/GeneralMaintenancePolicy</a></li><li>• <a href="https://bugs.ruby-lang.org/issues/8835">https://bugs.ruby-lang.org/issues/8835</a></li></ul> <p>So, the questions are as follows:</p> <ul style="list-style-type: none"><li>• Can we adopt Semantic Versioning (or as much of it as possible) across Ruby?</li><li>• Would such a change help users of Ruby?</li><li>• Is there existing documentation about how version number works?</li><li>• How does it deviate from Semantic Versioning?</li><li>• Is this deviation important and worth the additional complexity for our users?</li></ul> <p>As an aside:</p> <ul style="list-style-type: none"><li>• How do other implementations advertise compatibility with Ruby?</li><li>• JRuby and RBX have totally different version numbers that are difficult to understand w.r.t. compatibility with mainline CRuby.</li></ul> <p>My main concern is how difficult this is for everyone to keep track of and also the implied assumptions (e.g. breaking change if and only if major versions bump). If different parts of Ruby use different versioning scheme, it is hard for our users to define dependencies which don't cause broken software.</p> |        |

#### History

##### #1 - 12/23/2018 11:02 PM - shevegen (Robert A. Heiler)

I was about to write a lot as a comment but I feel it just gets too long.

So a few comments - sorry for being short:

- You should not forget that management of gems (code in these gems) also takes time away from developers who maintain/keep ruby up to date.
- Gems, or code, that is distributed with ruby itself, should be of higher priority than code hosted on rubygems.org or elsewhere. I disagree with any other way around e. g. where rubygems would be able to induce changes onto ruby. I mention this specifically because in several of these github issues, it feels to me as if people ignore or forget that and I think this is bad. Not just in the links you provide above, but I also saw prior discussions here about other gems.

Ultimately I think the only thing that can be done here in the short term is to improve gems + rubygems.org.

- I agree with the consistency comment to some extent; in particular when gems in stdlib would break in major ways, that should probably lead to a situation where we could have multiple different versions, just as we have multiple different ruby versions. But this also brings us to the limitation of gems right now. We have only one name for a gem on rubygems.org (bundler allowed for more flexibility here with e. g. github-based projects); and we do not easily have multiple different versions available for different ruby versions or projects, in particular if some gems are

pulled. That brings me back to the comment where I think gem + rubygems.org should be improved in the long run and provide more flexibility.

I understand that this does not directly have that much to do with your issue about adopting a specific versioning scheme, but I think the versioning is actually a secondary issue, as long as we provide as much useful information as possible, while allowing users to use older gems too. But this also brings us back to changing gems so ...

- Last but not least, I think it may help what you specifically propose.

For example:

"breaking change if and only if major versions bump"

This is a perfectly fine suggestion, for those users who may be affected.

But I, for example, also WANT to be able to use more up to date code and I don't mind breaking changes IF I can decide what to use and what not to use; so I would not agree to your statement if it means that it were to restrict me.

I am all in favour of heavily improving the whole gem ecosystem though; some of this may have to come from core ruby and matz may have to consider any changes there past 3.0. For example, "ownership" of "namespaces" - I don't mean this in the sense of restricting what others can do (I would be against this), but by providing meta-information about any change made on top of duck patching any ruby code too. But I digress. (On a side note, I do not version my gems, ironically enough, largely because I got tired of gem telling me about my own gems and own code that I could not easily install it - I always use the latest version of my own code and gems, so I could not accept gems restricting me here.)

#### #2 - 12/24/2018 12:57 PM - naruse (Yui NARUSE)

You can understand Ruby versioning as some kind of rolling release. X.Y is decided with marketing consideration, though .Z is the same as Semantic Versioning's TEENY. see also <https://www.ruby-lang.org/en/news/2013/12/21/ruby-version-policy-changes-with-2-1-0/>

Additional to say, I'm against the idea around X.Y of Semantic Versioning. Through my experience including both CRuby development and business, major version bump itself cause incompatibility in bundler ecosystems.

Moreover what you said in <https://github.com/ruby/bigdecimal/issues/114> seems not API incompatibility defined in Semantic Versioning. Semantic Versioning says nothing about the application's dependency.

And you should propose suggestion with practical merit. These days many people specify versions in Gemfile like gem "some-libs", "< 2". Bumping casually Semantic Versioning breaks such Gemfile and gems dependency.

You should also check why we need to downgrade bundler to 1.17 and released RC2. <https://github.com/rubygems/rubygems/pull/2515>  
You need to learn how bumping major version cause problems before enforcing major version bump.

#### #3 - 12/24/2018 03:35 PM - JonRowe (Jon Rowe)

These days many people specify versions in Gemfile like gem "some-libs", "< 2".  
Bumping casually Semantic Versioning breaks such Gemfile and gems dependency.

More people use gem "some-libs", "~> 2.0.0" and having breaking changes in a minor or patch release break the applications dependant on them.

This the typical style for web apps because it makes security updates easy to apply without changing a Gemfile / gemspec.

The whole point of a major version bump is stop both of these styles from accidentally picking up a breaking change.

#### #4 - 12/24/2018 09:42 PM - ioquatix (Samuel Williams)

Just for a point of reference, I use "~> x.y" for all my dependencies, as, taking into consideration semantic versioning, this allows for the maximum range of support but excluding breaking changes (i.e. x must stay the same, but y can increase).

This is all really good discussion, let's keep the ideas flowing.

## #5 - 12/27/2018 12:09 AM - naruse (Yui NARUSE)

Just for a point of reference, I use "~> x.y" for all my dependencies, as, taking into consideration semantic versioning, this allows for the maximum range of support but excluding breaking changes (i.e. x must stay the same, but y can increase).

Yeah, as `lib/rubygems/version.rb` says people should use "~> x.y" to avoid breaking changes if you are pessimistic.

```
# Specification From ... To (exclusive)
# ">= 3.0"      3.0 ... &infin;
# "~> 3.0"      3.0 ... 4.0
# "~> 3.0.0"    3.0.0 ... 3.1
# "~> 3.5"      3.5 ... 4.0
# "~> 3.5.0"    3.5.0 ... 3.6
# "~> 3"        3.0 ... 4.0
```

But dropping old rubies will happen every year and it can affect gems to remove legacy code.

If it is considered as breaking changes as you said, it will cause major bump and it's false positive for normal users who runs application with current rubies.

I think such false positive hearts Semantic Versioning.

## #6 - 12/27/2018 12:17 AM - ioquatix (Samuel Williams)

But dropping old rubies will happen every year and it can affect gems to remove legacy code. If it is considered as breaking changes as you said, it will cause major bump and it's false positive for normal users who runs application with current rubies.

I'm sorry, but I don't understand how this is a problem. Can you explain it in more detail with an example?

## #7 - 01/10/2019 07:19 AM - naruse (Yui NARUSE)

ioquatix (Samuel Williams) wrote:

But dropping old rubies will happen every year and it can affect gems to remove legacy code. If it is considered as breaking changes as you said, it will cause major bump and it's false positive for normal users who runs application with current rubies.

I'm sorry, but I don't understand how this is a problem. Can you explain it in more detail with an example?

Mr.A create `a1.gem` and its version is 1.0 which support Ruby 2.0, 2.1, and 2.2.

Mr.B write an application which uses `a1.gem` and write in gem "a1", "~> 1.0" Gemfile.

Next year, Mr.A drops Ruby 2.0 support and released `a1.gem` as 2.0.

Mr.B update Gemfile as gem "a1", "~> 2.0".

Next year, Mr.A drops Ruby 2.1 support and released `a1.gem` as 3.0.

Mr.B update Gemfile as gem "a1", "~> 3.0".

Next year, Mr.A drops Ruby 2.2 support and released `a1.gem` as 4.0.

Mr.B update Gemfile as gem "a1", "~> 4.0".

Next year, Mr.A drops Ruby 2.3 support and released `a1.gem` as 5.0.

Mr.B update Gemfile as gem "a1", "~> 5.0".

This is yearly normal changes is syntactically equal to big bang change.

## #8 - 01/26/2019 11:05 AM - ioquatix (Samuel Williams)

My understanding of semantic versioning is that what you depend on is not part of your public API. So, if you drop versions of Ruby, you can release new minor version, it's good enough.

## #9 - 01/26/2019 10:05 PM - Eregon (Benoit Daloze)

Just one example where a popular gem was basically forced to keep compatibility with older versions, or needs a major version increase:

<https://github.com/ruby-concurrency/concurrent-ruby/issues/768>

I'd guess there are many other similar cases.

One concern there is gems depending on the gem removing compatibility are forced to drop compatibility for that Ruby version too, or have very strict version constraints (such as `== last_working_version`).

**#10 - 01/26/2019 11:25 PM - MSP-Greg (Greg L)**

I'm confused. I'll use major.minor.teeny

Next year, Mr.A drops Ruby 2.0 support and released a1.gem as 2.0.  
Mr.B update Gemfile as gem "a1", "~> 2.0".

Next year, Mr.A drops Ruby 2.1 support and released a1.gem as 3.0.  
Mr.B update Gemfile as gem "a1", "~> 3.0".

No mention as to whether the releases contain breaking API changes. If not, why a new 'major' release, vs a 'minor' release?

Or, if the only change is dropping compatibility with a Ruby version, why is a new major release required?

**#11 - 01/26/2019 11:43 PM - MSP-Greg (Greg L)**

Ok, maybe I'm not confused.

Assume gem a-1.8 exists with a lower Ruby version constraint of  $\geq 2.0$ , and a-1.9 exists with a lower Ruby version constraint of  $\geq 2.2$ .

If one has pessimistic versioning of major.minor, RubyGems will select 1.9, but won't be able to install it on Ruby 2.0.

This is due to the fact that RubyGems improperly handles Ruby version constraints. IMO, this issue discourages the use of newer Ruby versions and also newer gem versions. It should be fixed, and it should be added to new releases of both RubyGems 2.x and 3.x.

**#12 - 02/07/2019 10:51 AM - ioquatix (Samuel Williams)**

[Eregon \(Benoit Daloze\)](#) If users are on 1.9.3, and the gem 1.0.5 supported it, and 1.1.0 didn't, I think you can argue that (a) it violates semantic versioning because a minor version bump broke the build but you could also argue that (b) it's compatible with semantic versioning because the ruby dependency is an internal detail to the gem (at least from it's own POV).

That being said, users who say gem 'concurrent-ruby', '~> 1.0' should correctly receive the latest compatible gem which suits their Ruby version, and that seems fine to me. The fact it's not being selected correctly might be a bug as [MSP-Greg \(Greg L\)](#) seems to be pointing out?

Looking at the bigger picture, I see that there are two cases, highlighted by the linked issue: pain for old users, or pain for up to date users. Honestly, I don't care so much about users of old releases. I do believe it should keep working, but I'm more concerned with breaking users who are using the latest stable release.  - so I think it was a mistake for concurrent-ruby to yank the 1.1.0 release, that caused real issues with production deployments.

**#13 - 02/07/2019 06:26 PM - Eregon (Benoit Daloze)**

ioquatix (Samuel Williams) wrote:

so I think it was a mistake for concurrent-ruby to yank the 1.1.0 release, that caused real issues with production deployments.

Right, especially since nowadays gem yank even deletes the file on the RubyGems server, instead of just skipping the version for Bundler resolution.

But it does mean to drop 1.9 support, concurrent-ruby can only do so with a 2.0 release to keep '~> 1' users happy.

MSP-Greg (Greg L) wrote:

Assume gem a-1.8 exists with a lower Ruby version constraint of  $\geq 2.0$ , and a-1.9 exists with a lower Ruby version constraint of  $\geq 2.2$ .

If one has pessimistic versioning of major.minor, RubyGems will select 1.9, but won't be able to install it on Ruby 2.0.

This is due to the fact that RubyGems improperly handles Ruby version constraints. IMO, this issue discourages the use of newer Ruby versions and also newer gem versions. It should be fixed, and it should be added to new releases of both RubyGems 2.x and 3.x.

Do you mean that the ruby version constraint of the gems should be taken into account for version resolution, and currently it's not at all?

This seems an interesting path forward: only consider versions of a gem which are compatible with the Ruby version running RubyGems or Bundler (or ruby "2.x.y" in the Gemfile).