PLEP-0005 – PlasmaPy Versioning and Releases

Nicholas A. Murphy and Stuart J. Mumford

PLEP	number
title	PlasmaPy Versioning and Releases
author(s)	Nick Murphy and Stuart Mumford
contact email	namurphy@cfa.harvard.edu
date created	2017-10-14
date last revised	2018-09-17
type	process
status	in preparation

Brief Description

PlasmaPy final releases will be of the form `MAJOR.MINOR.PATCH` where `MAJOR`, `MINOR`, and `PATCH` are non-negative integers that are always present. During the development phase (when `MAJOR` equals zero), the public application programming interface (API) should be considered unstable between minor releases. The first development release will be in early 2018, with subsequent development releases occurring every six months. Version `1.0.0` will be released after the API has stabilized and PlasmaPy is ready for production use. Starting with version `1.0.0`, `MAJOR` will be incremented when there are changes that remove backwards compatibility, `MINOR` will be incremented when backwards compatible functionality is added, and `PATCH` will be incremented for bug fixes and other changes that do not affect the API. A major or minor release should occur no less frequently than every six months, with major or long term support releases happening every other year.

Versioning Specification

PlasmaPy shall have version numbers that are consistent with the Semantic Versioning Specification 2.0.0appendix-semantic-versioning-specification-semver-200 and PEP 440.

Version numbers for PlasmaPy shall be of the following format:

`MAJOR.MINOR.PATCH[{a|b|rc}N][.devM]`

The `MAJOR`, `MINOR`, and `PATCH` version numbers must always be present. The pre-release tags given in square brackets are optional, and are absent for final releases which shall be of the form `MAJOR.MINOR.PATCH`. The pre-release tags are `a` for an alpha version, `b` for a beta version, and `rc` for a release candidate version. The integer `N` shall initially be `0`. The development tag `.devM` will be appended to versions undergoing development. The integer `M` shall be the commit number.

During the initial development phase, `MAJOR` will be `0` so final releases will have version numbers of the form `0.MINOR.PATCH`. `MINOR` shall be incremented whenever the public API changes, and `PATCH` shall be incremented whenever there are backwards compatible bug fixes. The API should be considered unstable during the development phase and anything may change at any time.

When the API has stabilized sufficiently across all subpackages and PlasmaPy has developed enough for widespread production use, version `1.0.0` shall be released. At this point, the following conditions will apply:

1. `MAJOR` must be incremented whenever we make incompatible changes to the application program interface (API).
2. MINOR must be incremented whenever we add backwards compatible functionality.
3. PATCH must be incremented whenever we make backwards compatible bug fixes.

Semantic Versioning Specification (SemVer) 2.0.0

This section contains the Semantic Version Specification 2.0.0 that was authored by Tom Preston-Werner and shared under a CC BY 3.0 license. PlasmaPy versioning will occur according to this standard.

> **Semantic Versioning 2.0.0**
>
> The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.
>
> 1. Software using Semantic Versioning MUST declare a public API. This API could be declared in the code itself or exist strictly in documentation. However it is done, it should be precise and comprehensive.
> 2. A normal version number MUST take the form X.Y.Z where X, Y, and Z are non-negative integers, and MUST NOT contain leading zeroes. X is the major version, Y is the minor version, and Z is the patch version. Each element MUST increase numerically. For instance: 1.9.0 -> 1.10.0 -> 1.11.0.

3. Once a versioned package has been released, the contents of that version MUST NOT be modified. Any modifications MUST be released as a new version.
4. Major version zero (0.y.z) is for initial development. Anything may change at any time. The public API should not be considered stable.
5. Version 1.0.0 defines the public API. The way in which the version number is incremented after this release is dependent on this public API and how it changes.
6. Patch version Z (x.y.Z | x > 0) MUST be incremented if only backwards compatible bug fixes are introduced. A bug fix is defined as an internal change that fixes incorrect behavior.
7. Minor version Y (x.Y.z | x > 0) MUST be incremented if new, backwards compatible functionality is introduced to the public API. It MUST be incremented if any public API functionality is marked as deprecated. It MAY be incremented if substantial new functionality or improvements are introduced within the private code. It MAY include patch level changes. Patch version MUST be reset to 0 when minor version is incremented.
8. Major version X (X.y.z | X > 0) MUST be incremented if any backwards incompatible changes are introduced to the public API. It MAY include minor and patch level changes. Patch and minor version MUST be reset to 0 when major version is incremented.
9. A pre-release version MAY be denoted by appending a hyphen and a series of dot separated identifiers immediately following the patch version. Identifiers MUST comprise only ASCII alphanumerics and hyphen [0-9A-Za-z-]. Identifiers MUST NOT be empty. Numeric identifiers MUST NOT include leading zeroes. Pre-release versions have a lower precedence than the associated normal version. A pre-release version indicates that the version is unstable and might not satisfy the intended compatibility requirements as denoted by its associated normal version. Examples: 1.0.0-alpha, 1.0.0-alpha.1, 1.0.0-0.3.7, 1.0.0-x.7.z.92.
10. Build metadata MAY be denoted by appending a plus sign and a series of dot separated identifiers immediately following the patch or pre-release version. Identifiers MUST comprise only ASCII alphanumerics and hyphen [0-9A-Za-z-]. Identifiers MUST NOT be empty. Build metadata SHOULD be ignored when determining version precedence. Thus two versions that differ only in the build metadata, have the same precedence. Examples: 1.0.0-alpha+001, 1.0.0+20130313144700, 1.0.0-beta+exp.sha.5114f85.
11. Precedence refers to how versions are compared to each other when ordered. Precedence MUST be calculated by separating the version into major, minor, patch and pre-release identifiers in that order (Build metadata does not figure into precedence). Precedence is determined by the first difference when comparing each of these identifiers from left to right as follows: Major, minor,

and patch versions are always compared numerically. Example: 1.0.0 < 2.0.0 < 2.1.0 < 2.1.1. When major, minor, and patch are equal, a pre-release version has lower precedence than a normal version. Example: 1.0.0-alpha < 1.0.0. Precedence for two pre-release versions with the same major, minor, and patch version MUST be determined by comparing each dot separated identifier from left to right until a difference is found as follows: identifiers consisting of only digits are compared numerically and identifiers with letters or hyphens are compared lexically in ASCII sort order. Numeric identifiers always have lower precedence than non-numeric identifiers. A larger set of pre-release fields has a higher precedence than a smaller set, if all of the preceding identifiers are equal. Example: 1.0.0-alpha < 1.0.0-alpha.1 < 1.0.0-alpha.beta < 1.0.0-beta < 1.0.0-beta.2 < 1.0.0-beta.11 < 1.0.0-rc.1 < 1.0.0.

Release Schedule

Version `0.1.0` of PlasmaPy was released in 2018 as a prototype and developer's preview. Subsequent development releases should occur no less frequently than every six months, and should occur more frequently when important new features are added. Minor releases during the development phase shall be supported with patch releases until the next minor release.

Version `1.0.0` will be released once PlasmaPy has a stable API that users have begun to depend upon. Releases should occur no less frequently than every six months. A major release should generally happen every two years. According to this schedule there should be about three minor releases between each major release.

Long term support (LTS) releases shall occur roughly once every two years. LTS releases shall be supported with maintenance and bug fix patches for at least two years or until the next LTS release, whichever takes longer. Version `1.0.0` should be PlasmaPy's first LTS release. Subsequent LTS releases should generally be the last minor release for each major version number. The Coordinating Committee may alter the LTS release schedule when appropriate (e.g., when major releases occur more or less frequently than every two years).

Issues, Pull Requests, and Branches

- https://github.com/PlasmaPy/PlasmaPy-PLEPs/pull/8
- Semantic Versioning: Why You Should Be Using It
- A critique of semantic versioning that proposes "romantic versioning"
- The SunPy community had a detailed conversation about switching to semantic versioning.

Alternatives

There are numerous versioning schemes that are used by different software projects. These schemes are generally less standardized between different projects than semantic versioning. Some options include:

- Version numbers may be of the form `YY.MM.PATCH` where `YY` corresponds to the last two digits of the year and `MM` corresponds to the digits associated with the month of the release. Ubuntu uses this versioning scheme. The advantage of this scheme is that it makes it easier to know when a version is out-of-date. A significant disadvantage is that this scheme provides no information on backwards compatibility.
- Some software packages have two versioning schemes. Public version numbers are easily human readable (e.g., by containing the year) whereas developers use a versioning scheme that provides more information about the state of development. This alternative is less useful for scientific packages where there is less distinction between users and developers.

Decision Rationale

Semantic versioning is a well-defined versioning scheme that provides users with useful information about whether or not there were any backward incompatible changes. This scheme is well-suited to a core scientific software package that will require stability.