versions Documentation

Release 0.2.0

Philippe Muller

December 14, 2013

Contents

Quick examples:

- Compare versions:

```
>>> from versions import Version
>>> Version.parse('2.0.0') > Version.parse('1.0.0')
True
```

- Test if constraints are satisfied by a version:

```
>>> from versions import Constraint, Constraints
>>> '2.0' in Constraint.parse('>1')
True
>>> '1.5' in Constraints.parse('>1,<2')
True
```

CHAPTER 1

Contents

1.1 Quickstart

1.1.1 Basic usage

Version comparison examples:

```
>>> from versions import Version
>>> v1 = Version.parse('1')
>>> v2 = Version.parse('2')
>>> v1 == v2
False
>>> v1 != v2
True
>>> v1 > v2
False
>>> v1 < v2
True
>>> v1 >= v2
False
>>> v1 <= v2
True
```

`Version.parse` expects a Semantic Version 2.0 string and returns a corresponding `Version` object:

```
>>> from versions import Version
>>> v = Version.parse('1.2.0-dev+foo.bar')
>>> v.major, v.minor, v.patch, v.prerelease, v.build_metadata
(1, 2, 0, 'dev', set(['foo', 'bar']))
```

If it isn't a semantic version string, the parser tries to normalize it:

```
>>> v = Version.parse('1')
>>> v.major, v.minor, v.patch, v.prerelease, v.build_metadata
(1, 0, 0, None, None)
```

1.1.2 Version constraint matching

versions also implements version constraint parsing and evaluation:

```
>>> from versions import Constraint
>>> Constraint.parse('>1').match('2')
True
>>> Constraint.parse('<2').match(Version.parse('1'))
True
```

For conveniance, constraint matching can be tested using the `in` operator:

```
>>> '1.5' in Constraint.parse('<2')
True
>>> Version('2') in Constraint.parse('!=2')
False
```

Constraints can be merged using `Constraints`:

```
>>> from versions import Constraints
>>> '1.0' in Constraints.parse('>1,<2')
False
>>> '1.5' in Constraints.parse('>1,<2')
True
>>> '2.0' in Constraints.parse('>1,<2')
False
```

1.2 API

Modules:

1.2.1 version

`Version` objects are comparable with standard operators:

```
>>> from versions import Version
>>> v1 = Version(1)
>>> v2 = Version.parse('2')
>>> v1 == v2
False
>>> v1 != v2
True
>>> v1 > v2
False
>>> v1 < v2
True
>>> v1 >= v2
False
>>> v1 <= v2
True
```

Version

class versions.version.**Version**(*major*, *minor=0*, *patch=0*, *prerelease=None*, *build_metadata=None*)

A package version.

Parameters

- **major** (*int*) – Version major number
- **minor** (*int*) – Version minor number
- **patch** (*int*) – Version patch number
- **prerelease** (str, int or None) – Version prerelease
- **build_metadata** (None or set of str) – Version build metadata

This class constructor is usually not called directly. For version string parsing, see Version.parse.

classmethod parse(*version_string*)

Parses a version_string and returns a Version object:

```
>>> Version.parse('1.0.0') > Version.parse('0.1')
```

True

InvalidVersion

exception versions.version.**InvalidVersion**(*version*)

Raised when failing to parse a version.

version = None

The bogus version.

1.2.2 constraint

Examples:

```
>>> from versions import Constraint
>>> Constraint.parse('>1').match('2')
True
>>> Constraint.parse('<2').match(Version.parse('1'))
True
>>> '1.5' in Constraint.parse('== 1.0')
False
>>> '1.5' in Constraint.parse('> 1.0')
True
```

Constraint

class versions.constraint.**Constraint**(*operator*, *version*)

A constraint on a package version.

Parameters

- **operator** (Operator) – The constraint operator.
- **version** (Version) – The constraint version.

match (*version*)
Match `version` with the contraint.

Parameters **version** (version string or `Version`) – Version to match against the constraint.

Return type `True` if `version` satisfies the constraint, `False` if it doesn't.

classmethod parse (*constraint_string*)
Parses a `constraint` string and returns a `Constraint` object.

Raises `InvalidConstraint` when parsing fails.

InvalidConstraint

exception `versions.constraint.`**InvalidConstraint** (*constraint*)
Raised when failing to parse a `constraint`.

constraint = None
The bogus constraint.

1.2.3 constraints

`Constraint` objects can be merged using `Constraints`:

```
>>> from versions import Constraints, Constraint
>>> '1.0' in Constraints.parse('>1,<2')
False
>>> '1.5' in Constraints.parse('>1,<2')
True
>>> '2.0' in Constraints.parse('>1,<2')
False
>>> Constraints.parse('>1') + '<2' + '!=1.5'
Constraints.parse('>1.0.0,<2.0.0,!=1.5.0')
```

Constraints

class `versions.constraints.`**Constraints** (*constraints=None*)
A collection of `Contraint` objects.

match (*version*)
Match `version` with this collection of constraints.

Parameters **version** (version string or `Version`) – Version to match against this collection of constraints.

Return type `True` if `version` satisfies this collection of constraint, `False` if it doesn't.

classmethod parse (*constraints_string*)
Parses a `constraints_string` and returns a `Constraints` object.

ExclusiveConstraints

exception `versions.constraints.`**ExclusiveConstraints** (*constraint*, *constraints*)
Raised when cannot merge a new constraint with pre-existing constraints.

constraint = None
The conflicting constraint.

constraints = None
The constraints with which it conflicts.

1.2.4 operators

Operator

class versions.operators.**Operator**(*func*, *string*)
A package version constraint operator.

Parameters

- **func** (*callable*) – The operator callable.
- **string** (*str*) – The operator string representation.

classmethod parse(*string*)
Parses *string* and returns an Operator object.

Raises InvalidOperator If *string* is not a valid operator.

Valid operators are ==, !=, <, >, <=, and >=.

InvalidOperator

exception versions.operators.**InvalidOperator**(*operator*)
Raised when failing to parse an operator.

operator = None
The bogus operator.

1.2.5 errors

Error

exception versions.errors.**Error**
An error occurred in versions.

CHAPTER 2

Indices and tables

- *genindex*
- *search*