versions Documentation

Release 0.5.0

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

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 `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

Comparison

`Version` objects are comparable with standard operators:

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

Parsing

`Version` has a convenient *parse* static method to parse constrints strings into `Version` objects.

The parser does its best to normalize the passed in string into a Semantic Version 2.0 version:

```
>>> from versions import Version
>>> Version.parse('1')
Version.parse('1.0.0')
>>> Version.parse('1.0')
Version.parse('1.0.0')
>>> Version.parse('1.0.0')
Version.parse('1.0.0')
>>> Version.parse('1.0.0-dev')
Version.parse('1.0.0-dev')
>>> Version.parse('1.0.0+some.build.data')
Version.parse('1.0.0+build.data.some')
>>> Version.parse('1.0.0-alpha+some.build.data')
Version.parse('1.0.0-alpha+build.data.some')
>>> Version.parse('1.0.0-42')
Version.parse('1.0.0-42')
```

InvalidVersion

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

Raised when failing to parse a `version`.

`version` = **None**

The bogus version.

1.2.2 constraint

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.

Parsing

`Constraint` has a convenient *parse* static method to parse constrints strings into `Constraint` objects.

Constraint strings are composed of a constraint operator, followed by a valid version string.

Valid constraint operators: ==, !=, <, >, <= and >=.

Matching

Examples:

```
>>> from versions import Constraint, Version
>>> Constraint.parse('>1').match('2')
True
>>> Constraint.parse('<2').match(Version.parse('1'))
True
>>> '1.5' in Constraint.parse('== 1.0')
False
>>> Version(1, 5) in Constraint.parse('> 1.0')
True
>>> Version(1) in Constraint.parse('>= 2.0.0')
False
```

InvalidConstraint

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

`constraint` = None
The bogus constraint.

1.2.3 constraints

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.

Merging

Constraint objects can be merged using a Constraints object and the + operator:

```
>>> from versions import Constraints, Constraint
>>> Constraints() + Constraint.parse('<2') + Constraint.parse('!=1.5')
Constraints.parse('<2.0.0,!=1.5.0')
```

Note: The Constraints object must be on the left side of the + operator. The Constraint object must be on right side.

If the constraint is a string, it is automatically parsed into a Constraint object. So the previous example can be shortened as:

```
>>> Constraints() + '<2' + '!=1.5'
Constraints.parse('<2.0.0,!=1.5.0')
```

Matching

Constraints objects work like Constraint objects: they have a *match()* method which returns True when passed a Version matching all constraints:

```
>>> Constraints.parse('>=1,<2').match('1.4')
True
>>> '1.4' in Constraints.parse('>=1.2,<2,!=1.4')
False
```

Conflicts

When merging conflicting constraints, an ExclusiveConstraints exception is raised:

```
>>> Constraints.parse('<1') + '>1'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/pmuller/dev/versions/versions/constraints.py", line 102, in __add__
    return Constraints(merge(self.constraints + [constraint]))
```

```
  File "/Users/pmuller/dev/versions/versions/constraints.py", line 209, in merge
    raise ExclusiveConstraints(g_constraint, [l_constraint])
versions.constraints.ExclusiveConstraints: Constraint >1.0.0 conflicts with constraints <1.0.0
>>> Constraints.parse('<1') + '==1'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/pmuller/dev/versions/versions/constraints.py", line 102, in __add__
    return Constraints(merge(self.constraints + [constraint]))
  File "/Users/pmuller/dev/versions/versions/constraints.py", line 223, in merge
    raise ExclusiveConstraints(eq_constraint, conflict_list)
versions.constraints.ExclusiveConstraints: Constraint ==1.0.0 conflicts with constraints <1.0.0
>>> Constraints.parse('>=1') + '!=1' + '<=1'
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/pmuller/dev/versions/versions/constraints.py", line 102, in __add__
    return Constraints(merge(self.constraints + [constraint]))
  File "/Users/pmuller/dev/versions/versions/constraints.py", line 223, in merge
    raise ExclusiveConstraints(eq_constraint, conflict_list)
versions.constraints.ExclusiveConstraints: Constraint ==1.0.0 conflicts with constraints !=1.0.0
```

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 packages

class versions.packages.**Package**(*name*, *version*)
A package.

Parameters

- **name** (*str*) – Package name.
- **version** (Version) – Package version.

name = None
Package name

version = None
Package version

1.2.6 errors

Error

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

CHAPTER 2

Indices and tables

- *genindex*
- *search*