appi Documentation

Release 0.1.7

Antoine Pinsard

Mar 24, 2018

Contents

Appi is a portage python interface. It is meant to be an alternative to the standard `portage` module, although, today, it is at a very early stage. It is designed to be straightforward to use, well-documented and have easy to read source code.

CHAPTER 1

Install appi

1.1 Installation from portage tree

First check if `dev-python/appi` is already in your portage tree:

```
emerge -av dev-python/appi
```

1.2 Installation from flora overlay

If your distribution does not provide a `dev-python/appi` ebuild, you can get it from the flora overlay:

```
mkdir -pv /var/overlays
git clone https://github.com/funtoo/flora.git /var/overlays/flora
cat > /etc/portage/repos.conf/flora <<EOF
[flora]
location = /var/overlays/flora
sync-type = git
sync-uri = git://github.com/funtoo/flora.git
auto-sync = yes
EOF
emerge -av dev-python/appi::flora
```

1.3 Installation from pypi

Not yet available.

1.4 Installation from git repository

```
pip install git+ssh://git@gitlab.com/apinsard/appi.git
```

CHAPTER 2

Get Started

Let's start python3, and write some `appi` calls:

```
$ python3
Python 3.4.5 (default, Nov 29 2016, 00:11:56)
[GCC 5.3.0] on linux
type "help", "copyright", "credits" or "license" for more information.
>>> import appi
>>>
```

2.1 Play with atoms

Something you must be familiar with are "query atoms", these are the strings used to query atoms with `emerge` and such tools. `appi.QueryAtom` is a class representing this kind of atoms, it enables you to check if a string is a valid atom or not.

Note: There is also a `DependAtom` which represents a dependency atom as found in ebuilds. It is not covered in this quick start but it behaves, to some extent, the same as `QueryAtom`.

2.1.1 Check atom validity

```
>>> appi.QueryAtom('dev-python/appi')
<QueryAtom: 'dev-python/appi'>
>>> appi.QueryAtom('=sys-apps/portage-2.4.3-r1')
<QueryAtom: '=sys-apps/portage-2.4.3-r1'>
>>> appi.QueryAtom('This is not a valid atom')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.4/site-packages/appi/atom.py", line 62, in __init__
    raise AtomError("{atom} is not a valid atom.", atom_string)
```

```
appi.atom.AtomError: This is not a valid atom is not a valid atom.
>>> from appi.exception import AtomError
>>> try:
...     appi.QueryAtom('>=dev-lang/python')
... except AtomError:
...     False
... else:
...     True
...
False
>>>
```

Note: `QueryAtom` only checks that the atom string is **valid**, not that an ebuild actually exists for this atom.

```
>>> appi.QueryAtom('this-package/does-not-exist')
<QueryAtom: 'this-package/does-not-exist'>
>>> appi.QueryAtom('~foo/bar-4.2.1')
<QueryAtom: '~foo/bar-4.2.1'>
>>>
```

Note: If you try to parse atoms without category name, you will notice that it raises an `AtomError` while it is actually a valid atom. There is a `strict` mode enabled by defaut, which makes package category mandatory in order to avoid dealing with ambiguous packages. You can easily disable this behavior by setting `strict=False`.

```
>>> appi.QueryAtom('=portage-2.4.3-r1')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.4/site-packages/appi/atom.py", line 71, in __init__
    atom_string, code='missing_category')
appi.atom.AtomError: =portage-2.4.3-r1 may be ambiguous, please specify the category.
>>> appi.QueryAtom('=portage-2.4.3-r1', strict=False)
<QueryAtom: '=portage-2.4.3-r1'>
>>>
```

2.1.2 Inspect atom parts

`QueryAtom` does not only check atoms validity, it also extracts its components.

```
>>> atom = appi.QueryAtom('=dev-lang/python-3*:3.4::gentoo')
>>> atom
<QueryAtom: '=dev-lang/python-3.4*:3.4::gentoo'>
>>> atom.selector
'='
>>> atom.category
'dev-lang'
>>> atom.package
'python'
>>> atom.version
'3'
>>> atom.postfix
'*'
>>> atom.slot
```

```
'3.4'
>>> atom.repository
'gentoo'
>>> atom2 = appi.QueryAtom('foo-bar/baz')
>>> atom2.selector
>>> atom2.version
>>> atom2.category
'foo-bar'
>>>
```

2.1.3 And much more!

Now, would you like to get the list of ebuilds that satisfy this atom? Nothing's easier!

```
>>> atom = appi.QueryAtom('=dev-lang/python-3*:3.4::gentoo')
>>> atom.list_matching_ebuilds()
{<Ebuild: 'dev-lang/python-3.4.3-r1::gentoo'>, <Ebuild: 'dev-lang/python-3.4.5::gentoo
↪'>}
>>>
```

Well, this brings us to ebuilds.

2.2 Go on with ebuilds

An `appi.Ebuild` instance represents the file describing a given version of a given package.

2.2.1 Check ebuild validity

Just as with atoms, you can check the validity of an ebuild by instantiating it.

```
>>> appi.Ebuild('/usr/portage/sys-devel/clang/clang-9999.ebuild')
<Ebuild: 'sys-devel/clang-9999::gentoo'>
>>> appi.Ebuild('/home/tony/Workspace/Funtoo/sapher-overlay/x11-wm/qtile/qtile-0.10.6.
↪ebuild')
<Ebuild: 'x11-wm/qtile-0.10.6::sapher'>
>>> appi.Ebuild('/usr/portage/sys-devel/clang/9999.ebuild')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.4/site-packages/appi/ebuild.py", line 58, in __init__
    raise EbuildError("{ebuild} is not a valid ebuild path.", path)
appi.ebuild.EbuildError: /usr/portage/sys-devel/clang/9999.ebuild is not a valid␣
↪ebuild path.
>>> from appi.exception import EbuildError
>>> try:
...     appi.Ebuild('/usr/portage/sys-devel/clang/clang-9999')
... except EbuildError:
...     False
... else:
...     True
...
False
>>> appi.Ebuild('/Unexisting/overlay/path/foo/bar/bar-1.5a_pre5-r12.ebuild')
```

```
<Ebuild: 'foo/bar-1.5a_pre5-r12'>
>>>
```

Warning: Note that currently, valid paths to unexisting files are considered valid ebuilds. This behavior is very likely to change as of version `0.1` since reading the ebuild file will be needed to extract some information such as slots and useflags. Thus, the `Ebuild` constructor may also raise `OSError` exceptions such as `FileNotFoundError` in future versions.

2.2.2 Inspect ebuild parts

```
>>> e = appi.Ebuild('/usr/portage/sci-libs/gdal/gdal-2.0.2-r2.ebuild')
>>> e.category
'sci-libs'
>>> e.package
'gdal'
>>> e.version
'2.0.2-r2'
>>> e.repository
<Repository: 'gentoo'>
>>> e.repository.location
PosixPath('/usr/portage')
>>>
```

2.2.3 And much more

You can check if an ebuild matches a given atom:

```
>>> e = appi.Ebuild('/usr/portage/app-portage/gentoolkit/gentoolkit-0.3.2-r1.ebuild')
>>> e.matches_atom(appi.QueryAtom('~app-portage/gentoolkit-0.3.2'))
True
>>> e.matches_atom(appi.QueryAtom('>gentoolkit-0.3.2', strict=False))
True
>>> e.matches_atom(appi.QueryAtom('>=app-portage/gentoolkit-1.2.3'))
False
>>> e.matches_atom(appi.QueryAtom('=app-portage/chuse-0.3.2-r1'))
False
>>>
```

2.3 Finally, let's checkout versions

Atom and ebuild objects both define a `get_version()` method that returns the version number as a `Version` object.

```
>>> atom = appi.QueryAtom('=x11-wm/qtile-0.10*')
>>> atom.version
'0.10'
>>> atom.get_version()
<Version: '0.10'>
>>> [(eb, eb.get_version()) for eb in atom.list_matching_ebuilds()]
```

```
[(<Ebuild: 'x11-wm/qtile-0.10.5::gentoo'>, <Version: '0.10.5'>), (<Ebuild: 'x11-wm/
↪qtile-0.10.6::gentoo'>, <Version: '0.10.6'>)]
>>>
```

2.3.1 Inspect version parts

```
>>> v = Version('3.14a_beta05_p4_alpha11-r16')
>>> v.base
'3.14'
>>> v.letter
'a'
>>> v.suffix
'_beta05_p4_alpha11'
>>> v.revision
'16'
>>>
```

2.3.2 Compare versions

```
>>> v1 = Version('2.76_alpha1_beta2_pre3_rc4_p5')
>>> v2 = Version('1999.05.05')
>>> v3 = Version('2-r5')
>>> v1 == v2
False
>>> v1 > v3
True
>>> v1 < v2
True
>>> v3.startswith(v1)
False
>>> Version('0.0a-r1').startswith(Version('0.0'))
True
>>>
```

CHAPTER 3

Reference

3.1 appi

3.1.1 appi.conf

appi.conf.Profile

A portage profile.

Currently, this only allows to retrieve the system `make.conf`. By "system", it is meant: after parsing `/usr/share/portage/config/make.globals`, profiles `make.defaults` and `/etc/portage/make.conf`.

Accross future versions, features will be implemented to retrieve all information contained in profiles, separately or all profiles aggregated.

Profile.list() -> list

Return the list of all enabled profiles, sorted in the order they will be parsed in the chain.

Examples

```
>>> Profile.list()
[<Profile: '/usr/portage/profiles/base'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/arch/base'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/arch/x86-
↪64bit'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/build/
↪current'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/arch/x86-
↪64bit/subarch/generic_64'>,
```

```
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/flavor/
↪minimal'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/flavor/core
↪'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/mix-ins/
↪console-extras'>,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/mix-ins/X'>
↪,
 <Profile: '/var/git/meta-repo/kits/core-kit/profiles/funtoo/1.0/linux-gnu/mix-ins/no-
↪systemd'>]
>>>
```

Profile.get_system_make_conf() -> dict

Return a dictionnary of system `make.conf` variables.

Examples

```
>>> a = Profile.get_system_make_conf()
>>> a['ARCH']
'amd64'
>>> a['KERNEL']
'linux'
>>> a['EMERGE_DEFAULT_OPTS']
'-j --load-average=5 --keep-going --autounmask=n'
>>> a['PORTAGE_TMPDIR']
'/var/tmp'
>>>
```

Profile(path)

Create a profile object from an absolute path. `path` must be a path to the directory describing the profile.

appi.conf.Repository

An ebuild repository.

Repository.get_main_repository() -> Repository

Return the main repository.

Repository.list(**kwargs) -> list

Return the list of repositories. Keyword arguments may be passed to filter repositories according to repository properties. Currently, the only accepted property is `location`.

Examples

```
>>> Repository.list(location='/var/git/meta-repo/kits/python-kit')
[<Repository: 'python-kit'>]
>>> Repository.list()
[<Repository: 'net-kit'>, <Repository: 'nokit'>, <Repository: 'core-hw-kit'>,
 <Repository: 'editors-kit'>, <Repository: 'games-kit'>, <Repository: 'python-kit'>,
 <Repository: 'gnome-kit'>, <Repository: 'java-kit'>, <Repository: 'media-kit'>,
 <Repository: 'perl-kit'>, <Repository: 'xorg-kit'>, <Repository: 'kde-kit'>,
 <Repository: 'core-kit'>, <Repository: 'security-kit'>, <Repository: 'php-kit'>,
 <Repository: 'dev-kit'>, <Repository: 'desktop-kit'>, <Repository: 'science-kit'>,
 <Repository: 'text-kit'>]
>>> Repository.list(location='/var/git/meta-repo/kits/python-kit')
[<Repository: 'python-kit'>]
>>>
```

Repository.find(**kwargs) -> Repository

Return the only repository that matches the passed keyword arguments. If no repository matches, return `None`. If more than one repository match, raises `ValueError`.

See also `Repository.list(**kwargs)`.

Raises

- `ValueError` if more than one repository match

Examples

```
>>> Repository.find(location='/var/git/meta-repo/kits/python-kit')
[<Repository: 'python-kit'>]
>>> Repository.find()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.4/site-packages/appi/conf/base.py", line 72, in find
    raise ValueError
ValueError
>>>
```

Repository.list_locations() -> generator

Return all repository locations.

3.1.2 appi.exception

appi.exception.AppiError

Appi base error. All errors thrown by appi will inherit this class.

`AppiError` will never be raised itself, its only goal is to serve as a catch-all exception for appi errors.

Any exception inheriting `AppiError` will have a `code` attribute giving more programmatically readable information about the error that occurred. See specific exceptions documentation to get a comprehensive list of applicable error codes.

Had errors have to be propagated to the user, error messages must be rendered by casting the exception to string.

appi.exception.AtomError

PortageError related to an atom.

Error codes

- `invalid` (default) - the atom format is invalid, no further details
- `missing_category` - expecting category because strict mode is enabled
- `missing_selector` - expecting a version selector because a version was specified
- `missing_version` - expecting a version because a version selector was specified
- `unexpected_revision` - a revision was given where it was not expected
- `unexpected_postfix` - a postfix was appended to the version while the version selector is not =
- `unexpected_slot_operator` - a slot operator was given where it was not expected

appi.exception.EbuildError

PortageError related to an ebuild.

Error codes

- `invalid` (default) - invalid ebuild path, no further details
- `package_name_mismatch` - mismatch between the package name described by the ebuild directory and the one described by the ebuild filename

appi.exception.PortageError

AppiError related to portage.

This is a catch-all exception for portage-related appi errors. It will never be raised itself.

appi.exception.VersionError

PortageError related to a package version.

Error codes

- `invalid` (default) - the version number is invalid, no further details

3.1.3 appi.DependAtom

A "depend atom" is a antom that is used in the DEPEND ebuild variable.

Its usage is very close to *QueryAtom* to some extent:

- It **can** be prefixed with ! or !!.
- It **cannot** be restricted to a specific repository (::repo).
- It **can** be appended a comma-separated list of useflags between brackets.

3.1.4 appi.Ebuild

Ebuild(path)

Create an ebuild object from an absolute path. path must be a valid ebuild path. A valid ebuild path starts with the repository location, then a category directory, a package directory and a package/version file with .ebuild extension.

Raises

- *EbuildError* if path is not a valid ebuild path.

Examples

```
>>> appi.Ebuild('/usr/portage/x11-wm/qtile/qtile-0.10.6.ebuild')
<Ebuild 'x11-wm/qtile-0.10.6::gentoo'>
>>> appi.Ebuild('/home/tony/Workspace/Funtoo/sapher-overlay/x11-wm/qtile/qtile-0.10.6.
↪ebuild')
<Ebuild 'x11-wm/qtile-0.10.6::sapher'>
>>> appi.Ebuild('/undefined/x11-wm/qtile/qtile-0.10.6.ebuild')
<Ebuild 'x11-wm/qtile-0.10.6'>
>>> appi.Ebuild('/x11-wm/qtile/qtile-0.10.6.ebuild')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib/python3.4/site-packages/appi/ebuild.py", line 59, in __init__
    raise EbuildError("{ebuild} is not a valid ebuild path.", path)
appi.ebuild.EbuildError: /x11-wm/qtile/qtile-0.10.6.ebuild is not a valid ebuild path.
>>>
```

Attributes

- **category** (str) The package category
- **package** (str) The package name
- **version** (str) The package version
- **repository** (*appi.conf.Repository*) The package repository if available, None otherwise
- **location** (pathlib.Path) The path of the ebuild file
- **useflags** (set) The set of useflags supported by this ebuild
- **slot** (str) The slot of the package

- **subslot** (`str`) The subslot of the package if any, `None` otherwise
- **vars** (`dict`) A dictionnary containing ebuild raw variables such as `HOMEPAGE`, `LICENSE`, `DESCRIPTION` and `EAPI`
- **db_dir** (`pathlib.Path`) The directory where information about this package installation can be found (if it is installed)

Examples

```
>>> e = appi.Ebuild('/usr/portage/app-editors/vim-core/vim-core-8.0.0386.ebuild')
>>> e.category
'app-editors'
>>> e.package
'vim-core'
>>> e.version
'8.0.0386'
>>> e.repository
<Repository 'gentoo'>
>>> e.useflags
{'acl', 'minimal', 'nls'}
>>> e.slot
'0'
>>> e.subslot
>>> f = appi.Ebuild('/tmp/app-editors/vim-core/vim-core-8.0.0386.ebuild')
>>> f.repository
>>> g = appi.Ebuild('/usr/portage/dev-lang/python/python-3.5.3.ebuild')
>>> g.slot
'3.5'
>>> g.subslot
'3.5m'
>>> g.vars['LICENSE']
'PSF-2'
>>>
```

String representation

The string representation of an ebuild is as following: `<category>/<name>-<version>`. Also, if the repository is known, it is appended as `::<repository>`.

Examples

```
>>> str(appi.Ebuild('/usr/portage/dev-python/appi/appi-0.0.ebuild'))
'dev-python/appi-0.0::gentoo'
>>> str(appi.Ebuild('/home/tony/Workspace/Funtoo/sapher-overlay/dev-python/appi/appi-
↪1.0.ebuild'))
'dev-python/appi-1.0::sapher'
>>> str(appi.Ebuild('/not/a/repository/dev-python/appi/appi-0.1.ebuild')
'dev-python/appi-0.1'
>>>
```

get_version() -> appi.Version

`Ebuild.version` is a string representing the version of the ebuild. `get_version()` returns it as a *Version* object.

Examples

```
>>> e = appi.Ebuild('/usr/portage/media-libs/libcaca/libcaca-0.99_beta19.ebuild')
>>> e.version
'0.99_beta19'
>>> e.get_version()
<Version '0.99_beta19'>
```

matches_atom(atom) -> bool

Return `True` if the ebuild matches the given `atom`.

Examples

```
>>> e = appi.Ebuild('/usr/portage/media-gfx/blender/blender-2.72b-r4.ebuild')
>>> e.matches_atom(appi.QueryAtom('=media-gfx/blender-2.72b-r4'))
True
>>> e.matches_atom(appi.QueryAtom('media-gfx/gimp'))
False
>>> e.matches_atom(appi.QueryAtom('~media-gfx/blender-2.72b'))
True
>>> e.matches_atom(appi.QueryAtom('>media-gfx/blender-2.72'))
True
>>> e.matches_atom(appi.QueryAtom('<=media-gfx/blender-2.72'))
False
>>> e.matches_atom(appi.QueryAtom('=media-gfx/blender-2*'))
True
>>> f = appi.Ebuild('/usr/portage/dev-lang/python/python-3.4.5.ebuild')
>>> f.matches_atom(appi.QueryAtom('dev-lang/python:3.4/3.4m')
True
>>> f.matches_atom(appi.QueryAtom('dev-lang/python:3.4')
True
>>> f.matches_atom(appi.QueryAtom('dev-lang/python:3.5')
False
>>>
```

is_installed() -> bool

Return `True` if this ebuild is installed. `False` otherwise.

3.1.5 appi.QueryAtom

A "query atom" is an atom that is used for querying a package, this is the kind of atoms accepted by `emerge` for instance.

There also exist *DependAtom* that have a slightly different format and is used by `DEPEND` variables in ebuilds.

QueryAtom(atom_string, strict=True)

Create a query atom from its string representation. `atom_string` must be a valid string representation of a query atom. The `strict` argument controls the "strict mode" state. When strict mode is enabled, the package category is mandatory and an error will be raised if it is missing. When strict mode is disabled, the package category is optional, which makes, for instance, `=firefox-50-r1` a valid atom.

Raises

- *AtomError* if `atom_string` is not a valid atom taking `strict` mode into consideration. Possible error codes:
  - **missing_category** The package category is missing, this can be ignored by setting `strict=False`.
  - **missing_selector** The package version was specified but the version selector is missing.
  - **missing_version** The version selector was specified but the package version is missing.
  - **unexpected_revision** The version contains a revision number while the version selector is ~.
  - **unexpected_postfix** The * postfix is specified but the version selector is not =.

Examples

```
>>> appi.QueryAtom('>=www-client/firefox-51')
<QueryAtom: '>=www-client/firefox-51'>
>>> appi.QueryAtom('=www-client/chromium-57*')
<QueryAtom: '=www-client/chromium-57*'>
>>> appi.QueryAtom('www-client/lynx')
<QueryAtom: 'www-client/lynx'>
>>> appi.QueryAtom('=www-client/links')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.5/site-packages/appi/atom.py", line 79, in __init__
    code='missing_version')
appi.atom.AtomError: =www-client/links misses a version number.
>>> appi.QueryAtom('google-chrome')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.5/site-packages/appi/atom.py", line 71, in __init__
    atom_string, code='missing_category')
appi.atom.AtomError: google-chrome may be ambiguous, please specify the category.
>>> appi.QueryAtom('google-chrome', strict=False)
<QueryAtom: 'google-chrome'>
```

Attributes

- **selector** (`str`) The package selector (>=, <=, <, =, > or ~)
- **category** (`str`) The package category
- **package** (`str`) The package name
- **version** (`str`) The package version
- **postfix** (`str`) The package postfix (* is the only possible value)
- **slot** (`str`) The package slot

- **repository** (`str`) The name of the repository

All these attribute, excepted **package**, are optional and may be `None`.

Examples

```
>>> a = appi.QueryAtom('=dev-db/mysql-5.6*:5.6::gentoo')
>>> a.selector
'='
>>> a.category
'dev-db'
>>> a.package
'mysql'
>>> a.version
'5.6'
>>> a.postfix
'*'
>>> a.slot
'5.6'
>>> a.repository
'gentoo'
>>> b = appi.QueryAtom('~postgresql-9.4', strict=False)
>>> b.selector
'~'
>>> b.category
>>> b.package
'postgresql'
>>> b.version
'9.4'
>>> b.postfix
>>> b.slot
>>> b.repository
>>>
```

String Representation

The string representation of an atom is the raw atom string itself: `<selector><category>/<package>-<version><postfix>:<slot>::<repository>`

Examples

```
>> str(appi.QueryAtom('dev-db/postgresql'))
'dev-db/postgresql'
>>> str(appi.QueryAtom('<dev-db/postgresql-9.6'))
'<dev-db/postgresql-9.6'
>>> str(appi.QueryAtom('>=dev-db/postgresql-8.4-r1::gentoo'))
'>=dev-db/postgresql-8.4-r1::gentoo'
>>> str(appi.QueryAtom('dev-db/postgresql:9.4'))
'dev-db/postgresql:9.4'
>>> a = appi.QueryAtom('=postgresql-9.4-r1', strict=False)
>>> str(a)
'=postgresql-9.4-r1'
>>> a.category = 'dev-db'
```

```
>>> str(a)
'=dev-db/postgresql-9.4-r1'
```

Warning: This can be useful to change the package category of an existing instance as above if you want to read atoms without requiring category and infer it afterwards if it is not a ambiguous.

However, it is not recommended to change other attributes values. Validity won't be checked and this can lead to incoherent atoms as illustrated below. We don't prevent attributes from being altered, we assume you are a sane minded developer who knows what he is doing.

```
>>> # /!\ DONT DO THIS /!\
>>> a.selector = ''
>>> str(a)
'dev-db/postgresql-9.4-r1'
>>> # Why would you anyway?
>>>
```

get_version() -> appi.Version

`QueryAtom.version` is a string representing the version included in the atom. `get_version()` returns it as a *Version* object.

Examples

```
>>> a = appi.QueryAtom('>=media-gfx/image-magick-7.0:0/7.0.4.3')
>>> a.version
'7.0'
>>> a.get_version()
<Version '7.0'>
```

get_repository() -> appi.conf.Repository

`QueryAtom.repository` is the name of the repository included in the atom. `get_repository()` returns the repository a *Repository* object.

This may be useful if you want to get the path or other data from the repository.

Examples

```
>>> a = appi.QueryAtom('app-portage/chuse::gentoo')
>>> a.repository
'gentoo'
>>> a.get_repository()
<Repository: 'gentoo'>
>>> a = appi.QueryAtom('app-portage/chuse::sapher')
>>> appi.QueryAtom('app-portage/chuse::sapher').get_repository()
<Repository: 'sapher'>
>>> appi.QueryAtom('app-portage/chuse::unexisting').get_repository()
>>>
```

list_matching_ebuilds() -> {appi.Ebuild, ...}

Returns the `set` of all ebuilds matching this atom.

Examples

```
>>> appi.QueryAtom('app-portage/chuse').list_matching_ebuilds()
{<Ebuild: 'app-portage/chuse-1.0.2::gentoo'>, <Ebuild: 'app-portage/chuse-1.1::gentoo
↪'>,
<Ebuild: 'app-portage/chuse-1.0.2::sapher'>, <Ebuild: 'app-portage/chuse-1.1::sapher'>
↪}
>>> appi.QueryAtom('app-portage/chuse::gentoo').list_matching_ebuilds()
{<Ebuild: 'app-portage/chuse-1.0.2::gentoo'>, <Ebuild: 'app-portage/chuse-1.1::gentoo
↪'>}
>>> appi.QueryAtom('screen', strict=False).list_matching_ebuilds()
{<Ebuild: 'app-misc/screen-4.0.3-r9::funtoo-overlay'>, <Ebuild: 'app-vim/screen-1.
↪5::gentoo'>,
<Ebuild: 'app-misc/screen-4.0.3-r9::gentoo'>, <Ebuild: 'app-misc/screen-4.4.0::funtoo-
↪overlay'>,
<Ebuild: 'app-misc/screen-4.0.3-r10::funtoo-overlay'>,
<Ebuild: 'app-misc/screen-4.2.1-r2::funtoo-overlay'>, <Ebuild: 'app-misc/screen-4.4.
↪0::gentoo'>,
<Ebuild: 'app-misc/screen-4.0.3-r10::gentoo'>, <Ebuild: 'app-misc/screen-4.0.3-
↪r3::gentoo'>,
<Ebuild: 'app-misc/screen-4.0.3-r3::funtoo-overlay'>, <Ebuild: 'app-misc/screen-4.2.1-
↪r2::gentoo'>}
>>> appi.QueryAtom('<screen-4', strict=False).list_matching_ebuilds()
{<Ebuild: 'app-vim/screen-1.5::gentoo'>}
>>> appi.QueryAtom('<screen-1', strict=False).list_matching_ebuilds()
set()
>>> appi.QueryAtom('dev-lang/python:3.4::gentoo').list_matching_ebuilds()
{<Ebuild: 'dev-lang/python-3.4.5::gentoo'>, <Ebuild: 'dev-lang/python-3.4.6::gentoo'>}
>>>
```

matches_existing_ebuild() -> `bool`

Returns `True` if any existing ebuild matches this atom. `False` otherwise. Basically, it checks if `list_matching_ebuilds()` returns an empty set or not.

Examples

```
>>> appi.QueryAtom('dev-python/unexisting-module').matches_existing_ebuild()
False
>>> appi.QueryAtom('dev-python/appi').matches_existing_ebuild()
True
>>> appi.QueryAtom('~dev-python/appi-1.2.3').matches_existing_ebuild()
False
>>> appi.QueryAtom('screen', strict=False).matches_existing_ebuild()
True
>>>
```

is_installed() -> bool

Return `True` if any of the matching ebuilds is installed. `False` otherwise.

3.1.6 appi.Version

The `Version` object is the representation of a package version. It enables to compare versions.

Version(version_string)

Create a version object from a valid version string.

Raises

- *VersionError* if `version_string` is not a valid version number

Examples

```
>>> appi.Version('1.3')
<Version: '1.3'>
>>> appi.Version('3.14-r1')
<Version: '3.14-r1'>
>>> appi.Version('1.2.3a_rc4_pre5_alpha2-r6')
<Version: '1.2.3a_rc4_pre5_alpha2-r6'>
>>> appi.Version('2.0beta')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.5/site-packages/appi/version.py", line 76, in __init__
    "{version} is not a valid version.", version_string)
appi.version.VersionError: 2.0beta is not a valid version.
>>> appi.Version('2.0_beta')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.5/site-packages/appi/version.py", line 76, in __init__
    "{version} is not a valid version.", version_string)
appi.version.VersionError: 2.0_beta is not a valid version.
>>> appi.Version('bonjour')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/usr/lib64/python3.5/site-packages/appi/version.py", line 76, in __init__
    "{version} is not a valid version.", version_string)
appi.version.VersionError: bonjour is not a valid version.
>>>
```

Attributes

- **base** (`str`) The base version number (part before the letter if any)
- **letter** (`str`) The letter version number (a single letter), optional
- **suffix** (`str`) The suffix version number (release, pre-release, patch, ...), optional
- **revision** (`str`) The ebuild revision number, optional

Examples

```
>>> v = Version('1.2.3d_rc5_p0-r6')
>>> v.base
'1.2.3'
>>> v.letter
'd'
>>> v.suffix
'_rc5_p0'
>>> v.revision
'6'
>>>
```

compare(other) -> `int`

startswith(version) -> `bool`

get_upstream_version() -> `Version`