# vcstools Documentation

***Release 0.1***

**2010, Willow Garage**

September 26, 2011

# CONTENTS

The `vcstools` module provides a Python API for interacting with different version control systems (VCS/SCMs). The `VcsClient` class provides an API for seamless interacting with Git, Mercurial (Hg), Bzr and SVN. The focus of the API is manipulating on-disk checkouts of source-controlled trees. Its main use is to support the *rosinstall* tool.

CHAPTER

ONE

# GENERAL VCS/SCM API

The `VcsClient` class provides a generic API for

- Subversion (`svn`)
- Mercurial (`hg`)
- Git (`git`)
- Bazaar (`bzr`)

**class** `vcstools.`**`VcsClient`**(*vcs_type*, *path*)

API for interacting with source-controlled paths independent of actual version-control implementation.

**Parameters**

- **vcs_type** – type of VCS to use (e.g. 'svn', 'hg', 'bzr', 'git'), `str`
- **path** – filesystem path where code is/will be checked out , `str`

**`path_exists`**() → bool

**Returns** True if path exists on disk.

**`get_path`**() → str

**Returns** filesystem path this client is initialized with.

**`get_version`**([*spec=None* ])

**Parameters** **spec** – token for identifying repository revision desired. Token might be a tagname, branchname, version-id, or SHA-ID depending on the VCS implementation.

- svn: anything accepted by `svn info --help`, e.g. a `revnumber`, `{date}`, `HEAD`, `BASE`, `PREV`, or `COMMITTED`
- git: anything accepted by `git log`, e.g. a tagname, branchname, or sha-id.
- hg: anything accepted by `hg log -r`, e.g. a tagname, sha-ID, revision-number
- bzr: revisionspec as returned by `bzr help revisionspec`, e.g. a tagname or `revno:<number>`

**Returns** current revision number of the repository. Or if spec is provided, the globally unique identifier (e.g. revision number, or SHA-ID) of a revision specified by some token.

**`checkout`**(*url*[, *version=''* ])

Checkout the given URL to the path associated with this client.

**Parameters**

- **url** – URL of source control to check out

- **version** – specific version to check out

**update**(*version*)
Update the local checkout from upstream source control.

**detect_presence**() → bool

**Returns** True if path has a checkout with matching VCS type, e.g. if the type of this client is 'svn', the checkout at the path is managed by Subversion.

**get_vcs_type_name**() → str

**Returns** type of VCS this client is initialized with.

**get_url**() → str

**Returns** Upstream URL that this code was checked out from.

**get_branch_parent**()
(Git Only)

**Returns** parent branch.

**get_diff**([*basepath=None*])

**Parameters** **basepath** – compute diff relative to this path, if provided

**Returns** A string showing local differences

**get_status**([*basepath=None*[, *untracked=False*]])
Calls scm status command. semantics of untracked are difficult to generalize. In SVN, this would be new files only. In git, hg, bzr, this would be changes that have not been added for commit.

**Parameters**

- **basepath** – status path will be relative to this, if provided.
- **untracked** – If True, also show changes that would not commit

**Returns** A string summarizing locally modified files

Example:

```
import vcstools

# interrogate an existing tree
client = vcstools.VcsClient('svn', '/path/to/checkout')

print client.get_url()
print client.get_version()
print client.get_diff()

# create a new tree
client = vcstools.VcsClient('hg', '/path/to/new/checkout')
client.checkout('https://bitbucket.org/foo/bar')
```

CHAPTER TWO

# INSTALLATION

vcstools is available on pypi and can be installed via `pip`

```
pip install vcstools
```

or `easy_install`:

```
easy_install vcstools
```

CHAPTER

THREE

# USING ROSPKG

The `vcstools` module is meant to be used as a normal Python module. After it has been installed, you can `import` it normally and do not need to declare as a ROS package dependency.

CHAPTER

FOUR

# ADVANCED: ROSPKG DEVELOPERS/CONTRIBUTORS

## 4.1 Developer's Guide

### 4.1.1 Bug reports and feature requests

- Submit a bug report
- Submit a feature request

### 4.1.2 Developer Setup

vcstools uses setuptools, which you will need to download and install in order to run the packaging. We use setuptools instead of distutils in order to be able use `setup()` keys like `install_requires`.

Configure your `PYTHONPATH`:

```
cd vcstools
. setup.sh
```

OR:

```
cd vcstools
python setup.py install
```

The first will prepend `vcstools/src` to your `PYTHONPATH`. The second will install vcstools into your dist/site-packages.

### 4.1.3 Testing

Install test dependencies

```
pip install nose
pip install mock
```

rospkg uses Python nose for testing, which is a fairly simple and straightfoward test framework. The vcstools mainly use `unittest` to construct test fixtures, but with nose you can also just write a function that starts with the name `test` and use normal `assert` statements.

vcstools also uses mock to create mocks for testing.

You can run the tests, including coverage, as follows:

```
cd vcstools
make test
```

### 4.1.4 Documentation

Sphinx is used to provide API documentation for vcstools. The documents are stored in the `doc` subdirectory.

CHAPTER

FIVE

# INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

# PYTHON MODULE INDEX