rospkg Documentation

Release 0.2.0

Ken Conley, Tully Foote

September 14, 2011

CONTENTS

The `rospkg` module provides basic utilities for querying information about ROS packages and stacks. There are several basic APIs: *ROS environment*, `RosPack`/`RosStack`, and *OS detection*. The environment APIs enable access to environment settings that defines the ROS package/stack filesystem configuration. The `RosPack` and `RosStack` APIs are similar to the `rospack` and `rosstack` command-line tools and provide information about dependency, location, and other package/stack metadata. The `Manifest` class provides access to a specific package/stack's manifest information.

CHAPTER

ONE

ROS PACKAGE ACCESS

The `RosPack` class provides APIs similar to the `rospack` command-line tool distributed with ROS. Like `rospack`, its provides information about package and stack dependency information, filesystem locations, and manifest access. The Python API is more efficient than shelling out to `rospack` as provides caching and other optimizations for repeated querying.

`rospkg.`**`MANIFEST_FILE`**
Name of package manifest file, i.e. 'manifest.xml'.

class `rospkg.`**`RosPack`**(*[ros_root=None[, ros_package_path=None]]*)
Query information about ROS packages on the local filesystem. This includes information about dependencies, retrieving stack `Manifest` instances, and determining the parent stack of a package.

`RosPack` can be initialized with the default environment, or its environment configuration can be overridden with alternate

`ROS_ROOT` and `ROS_PACKAGE_PATH` settings.

NOTE: for performance reasons, `RosPack` caches information about packages

Example:

```
rp = RosPack()
packages = rp.list_packages()
path = rp.get_path('rospy')
depends = rp.get_depends('roscpp')
depends1 = rp.get_depends('roscpp', implicit=False)
```

Parameters

- **ros_root** – override `ROS_ROOT`.
- **ros_package_path** – override `ROS_PACKAGE_PATH`. To specify no `ROS_PACKAGE_PATH`, use the empty string. An assignment of None will use the default path.

`get_ros_root`() → str
Get the `ROS_ROOT` configuration of this instance.

`get_ros_package_path`() → str
Get the `ROS_PACKAGE_PATH` configuration of this instance.

`ros_root`
Get the `ROS_ROOT` configuration of this instance. Read-only.

`ros_package_path`
Get the `ROS_PACKAGE_PATH` configuration of this instance. Read-only.

get_manifest(*name*) → Manifest
Get the `Manifest` of the specified package.

Parameters **name** – package name, `str`

Raises `InvalidManifest`

list() → [str]
List packages.

Returns complete list of package names in ROS environment

get_path(*name*) → str

Parameters **name** – package name, `str`

Returns filesystem path of package

Raises `ResourceNotFound`

get_depends(*name*[, *implicit=True*]) → [str]
Get explicit and implicit dependencies of a package.

Parameters

- **name** – package name, `str`
- **implicit** – include implicit (recursive) dependencies, `bool`

Returns list of names of dependencies.

Raises `InvalidManifest`

get_rosdeps(*package*[, *implicit=True*]) → [str]
Collect rosdeps of specified package into a dictionary.

Parameters

- **package** – package name, `str`
- **implicit** – include implicit (recursive) rosdeps, `bool`

Returns list of rosdep names.

stack_of(*package*) → str

Parameters **package** – package name, `str`

Returns name of stack that *package* is in, or `None` if *package* is not part of a stack

Raises `ResourceNotFound`: if *package* cannot be located

CHAPTER

TWO

ROS STACK ACCESS

The RosStack classes provides APIs similar to the rosstack command-line tool distributed with ROS. Like rosstack, it provides information about stack dependency information, filesystem locations, and manifest access. The Python API is more efficient than shelling out to rosstack as it provides caching and other optimizations for repeated querying.

rospkg.**STACK_FILE**
Name of stack manifest file, i.e. 'stack.xml'.

class rospkg.**RosStack**([*ros_root=None*[, *ros_package_path=None*]])
Query information about ROS stacks on the local filesystem. This includes information about dependencies, retrieving stack Manifest instances, and determining the contents of stacks.

RosStack can be initialized with the default environment, or its environment configuration can be overridden with alternate

ROS_ROOT and ROS_PACKAGE_PATH settings.

NOTE: for performance reasons, RosPack caches information about packages.

Parameters

- **ros_root** – (optional) override ROS_ROOT.
- **ros_package_path** – (optional) override

  ROS_PACKAGE_PATH. To specify no

  ROS_PACKAGE_PATH, use the empty string. An assignment of None will use the default path.

get_ros_root() → str
Get the ROS_ROOT configuration of this instance.

get_ros_package_path() → str
Get the ROS_PACKAGE_PATH configuration of this instance.

ros_root
Get the ROS_ROOT configuration of this instance. Read-only.

ros_package_path
Get the ROS_PACKAGE_PATH configuration of this instance. Read-only.

get_manifest(*name*) → Manifest
Get the Manifest of the specified package.

Parameters **name** – package name, str

Raises InvalidManifest

list() → [str]
List stacks.

Returns complete list of package names in ROS environment

get_path(*name*) → str

Parameters **name** – stack name, `str`

Returns filesystem path of stack

Raises `ResourceNotFound`

get_direct_depends(*name*) → [str]
Get the explicit dependencies of a stack.

Parameters **name** – stack name, `str`

Returns list of names of direct dependencies

Raises `ResourceNotFound`

Raises `InvalidManifest`

get_depends(*name*) → [str]
Get explicit and implicit dependencies of a stack.

Parameters **name** – stack name, `str`

Returns list of names of dependencies.

Raises `InvalidManifest`

packages_of(*stack*) → [str]

Returns name of packages that are part of stack

Raises `ResourceNotFound` if stack cannot be located

get_stack_version(*stack*) → str

Parameters **env** – override environment variable dictionary

Returns version number of stack, or None if stack is unversioned.

rospkg.**expand_to_packages**(*names, rospack, rosstack) -> ([str], [str]*)
Expand names into a list of packages. Names can either be of packages or stacks.

Parameters

- **names** – list of names of stacks or packages, `[str]`
- **rospack** – `RosPack` instance
- **rosstack** – `RosStack` instance

Returns ([packages], [not_found]). `expand_packages()` returns two lists. The first is of packages names. The second is a list of names for which no matching stack or package was found. Lists may have duplicates.

rospkg.**get_stack_version_by_dir**(*stack_dir*) → str
Get stack version where stack_dir points to root directory of stack.

Parameters **env** – override environment variable dictionary

Returns version number of stack, or None if stack is unversioned.

CHAPTER

THREE

ROS ENVIRONMENT VARIABLE ACCESS

The environment APIs of `rospkg` provide convenient access to ROS package-related environment variables, including methods that provide default values when environment variable overrides are not active.

ROS_PACKAGE_PATH
Name of `ROS_PACKAGE_PATH` environment variable.

ROS_ROOT
Name of `ROS_ROOT` environment variable.

ROS_HOME
Name of `ROS_HOME` environment variable.

ROS_LOG_DIR
Name of `ROS_LOG_DIR` environment variable.

ROS_TEST_RESULTS_DIR
Name of `ROS_TEST_RESULTS_DIR` environment variable.

get_ros_root([*env=None*]) → str
Get the current `ROS_ROOT`.

Parameters **env** – override environment dictionary

get_ros_package_path([*env=None*])
Get the current `ROS_PACKAGE_PATH`.

Parameters **env** – (optional) environment override.

get_ros_home([*env=None*]) → str
Get directory location of `.ros` directory (aka `ROS_HOME`). possible locations for this. The `ROS_LOG_DIR` environment variable has priority. If that is not set, then

`ROS_HOME`/log is used. If `ROS_HOME` is not set, `$HOME/.ros/log` is used.

Parameters **env** – override environment dictionary

Returns path to use use for log file directory

get_log_dir([*env=None*]) → str
Get directory to use for writing log files. There are multiple possible locations for this. The `ROS_LOG_DIR` environment variable has priority. If that is not set, then

`ROS_HOME`/log is used. If `ROS_HOME` is not set, `$HOME/.ros/log` is used.

Parameters **env** – override environment dictionary

Returns path to use use for log file directory

get_test_results_dir(*env=None*) → str

Get directory to use for writing test result files. There are multiple possible locations for this. The `ROS_TEST_RESULTS_DIR` environment variable has priority. If that is set, `ROS_TEST_RESULTS_DIR` is returned. If `ROS_TEST_RESULTS_DIR` is not set, then `ROS_HOME`/test_results is used. If

`ROS_HOME` is not set, `$HOME/.ros/test_results` is used.

Parameters **env** – override environment dictionary

Returns path to use use for log file directory

on_ros_path(*p*[, *env=None*]) → bool

Check to see if filesystem path is on paths specified in ROS environment (`ROS_ROOT`, `ROS_PACKAGE_PATH`).

Parameters **p** – path, `str`

Returns True if p is on the ROS path (ROS_ROOT, ROS_PACKAGE_PATH)

CHAPTER

FOUR

OS DETECTION

OS detection is a critical capability of many ROS tools in the ROS build toolchain. The `rospkg.os_detect` module provides and extendable library for detecting various operating systems. It is focused on detecting operating systems used with ROS.

Currently supported OSes:

- Arch Linux
- Cygwin
- Debian
- Fedora
- FreeBSD
- Gentoo
- Mint
- OS X
- Red Hat Linux
- Ubuntu

class `rospkg.os_detect.`**`OsNotDetected`**

Exception to indicate failure to detect operating system.

class `rospkg.os_detect.`**`OsDetect`**(*os_list*)

Detects the current operating system. This class will iterate over registered classes to lookup the active OS and version. The list of detectors can be overridden in the constructor; otherwise it will default to `OsDetector` classes provided by this library.

`default_os_list`

List of currently registered detectors. Must not be modified directly.

static **`register_default`**(*os_name*, *os_detector*)

Register detector to be used with all future instances of `OsDetect`. The new detector will have precedence over any previously registered detectors associated with *os_name*.

`detect_os`() → tuple

Returns (os_name, os_version, os_codename), `(str, str, str)`

Raises `OsNotDetected` if OS could not be detected

`get_detector`([*name*]) → OsDetector

Get detector used for specified OS name, or the detector for this OS if name is `None`.

Raises `KeyError`

add_detector(*name*, *detector*)
Add detector to list of detectors used by this instance. *detector* will override any previous detectors associated with *name*.
Parameters
- **name** – OS name that detector matches
- **detector** – `OsDetector` instance

get_os() → OsDetector
Get `OsDetector` for this operating system.
Raises `OsNotDetected` if OS could not be detected

get_name() → str
Returns Name of current operating system. See `OS_*` definitions in this module for possible values.
Raises `OsNotDetected` if OS could not be detected

get_version() → str
Returns Version of current operating system
Raises `OsNotDetected` if OS could not be detected

get_codename() → str
Returns Codename of current operating system if available, or empty string if OS does not provide codename.
Raises `OsNotDetected` if OS could not be detected

class `rospkg.os_detect.`**OsDetector**
Generic API for detecting a specific OS.

get_codename()

Returns codename for this OS. (ala Ubuntu Hardy Heron = "hardy"). If codenames are not available for this OS, return empty string.

Raises `OsNotDetected` if called on incorrect OS.

get_version()

Returns standardized version for this OS. (ala Ubuntu Hardy Heron = "8.04")

Raises `OsNotDetected` if called on incorrect OS.

is_os()

Returns if the specific OS which this class is designed to detect is present. Only one version of this class should return for any version.

4.1 OS name definitions

`rospkg.os_detect.`**OS_ARCH**
Name used for Arch Linux OS.

`rospkg.os_detect.`**OS_CYGWIN**
Name used for Cygwin OS.

`rospkg.os_detect.`**OS_DEBIAN**
Name used for Debian OS.

`rospkg.os_detect.`**OS_FREEBSD**
Name used for FreeBSD OS.

`rospkg.os_detect.OS_GENTOO`
Name used for Gentoo.

`rospkg.os_detect.OS_MINT`
Name used for Mint OS.

`rospkg.os_detect.OS_OPENSUSE`
Name used for OpenSUSE OS.

`rospkg.os_detect.OS_OSX`
Name used for OS X.

`rospkg.os_detect.OS_RHEL`
Name used for Red Hat Enterprise Linux.

`rospkg.os_detect.OS_UBUNTU`
Name used for Ubuntu OS.

4.2 Linux helper methods

`rospkg.os_detect.lsb_get_os()` → str
Linux: wrapper around lsb_release to get the current OS

`rospkg.os_detect.lsb_get_codename()` → str
Linux: wrapper around lsb_release to get the current OS codename

`rospkg.os_detect.lsb_get_version()` → str
Linux: wrapper around lsb_release to get the current OS version

`rospkg.os_detect.uname_get_machine()` → str
Linux: wrapper around uname to determine if OS is 64-bit

CHAPTER

FIVE

ROSDISTRO FILE LIBRARY

This submodule provides the `Distro` class, which provides an API for processing rosdistro files.

Table of Contents

5.1 Data model

The top level representation is a `Distro` instance, which contains `Variant` and `DistroStack` instances. `DistroStack` instances have a `VcsConfig` (`SvnConfig`, `GitConfig`, `BzrConfig`, `HgConfig`), which represents the source control information for the stack.:

```
Distro
  - Variant
  - DistroStack
     - VcsConfig
```

5.2 Exceptions

5.3 Utility functions

`rospkg.distro.`**`distro_uri`**(*distro_name*)

Get distro URI of main ROS distribution files.

Parameters **distro_name** – name of distro, e.g. 'diamondback'

Returns the SVN/HTTP URL of the specified distro. This function should only be used with the main distros.

`rospkg.distro.`**`load_distro`**(*source_uri*) → Distro

Load `Distro` instance from *source_uri*.

Example:

```
from rospkg.distro import load_distro, distro_uri
d = load_distro(distro_uri('electric'))
```

Parameters **source_uri** – source URI of distro file, or path to distro file. Filename has precedence in resolution.

Raises `InvalidDistro` if distro file is invalid

Raises `rospkg.ResourceNotFound` if file at *source_uri* is not found

`rospkg.distro.`**`expand_rule`**(*rule*, *stack_name*, *stack_ver*, *release_name*) → str

Replace variables in VCS config rule value with specified values

`rospkg.distro.`**`distro_to_rosinstall`**(*distro*, *branch*[, *variant_name=None*[, *implicit=True*[, *released_only=True*[, *anonymous=True*]]]])

Parameters

- **branch** – branch to convert for
- **variant_name** – if not None, only include stacks in the specified variant.
- **implicit** – if variant_name is provided, include full (recursive) dependencies of variant, default True
- **released_only** – only included released stacks, default True.
- **anonymous** – create for anonymous access rules

Raises `KeyError` if branch is invalid or if distro is mis-configured

5.4 Model

class `rospkg.distro.`**`DistroStack`**

Stores information about a stack release

Parameters

- **stack_name** – Name of stack
- **stack_version** – Version number of stack.
- **release_name** – name of distribution release. Necessary for rule expansion.
- **rules** – raw '_rules' data. Will be converted into appropriate vcs config instance.

class `rospkg.distro.`**`Variant`**(*variant_name*, *extends*, *stack_names*, *stack_names_implicit*)

A variant defines a specific set of stacks ("metapackage", in Debian parlance). For example, "base", "pr2". These variants can extend another variant.

Parameters

- **variant_name** – name of variant to load from distro file, `str`
- **stack_names_implicit** – full list of stacks implicitly included in this variant, `[str]`
- **raw_data** – raw rosdistro data for this variant

`get_stack_names`([*implicit=True*]) → [str]

Get list of all stack names in this variant.

Parameters **implicit** – If `True`, includes names of stacks in parent variants. Otherwise, include only stacks explicitly named in this variant. (default `True`).

stack_names
List of all stack names in this variant, including implicit stacks.

class `rospkg.distro.`**`Distro`**(*stacks*, *variants*, *release_name*, *version*, *raw_data*)
Store information in a rosdistro file.

Parameters

- **stacks** – dictionary mapping stack names to `DistroStack` instances
- **variants** – dictionary mapping variant names to `Variant` instances
- **release_name** – name of release, e.g. 'diamondback'
- **version** – version number of release
- **raw_data** – raw dictionary representation of a distro

get_stacks([*released=False*]) → {str: DistroStack}

Parameters **released** – only included released stacks

Returns dictionary of stack names to `DistroStack` instances in this distro.

5.5 Source control information

`rospkg.distro.`**`get_vcs_configs`**() → {str: VcsConfig}

Returns Dictionary of supported `VcsConfig` instances. Key is the VCS type name, e.g. 'svn'.

`rospkg.distro.`**`load_vcs_config`**(*rules*, *rule_eval*) → VcsConfig
Factory for creating `VcsConfig` subclass based on rosdistro _rules data.

Parameters

- **rules** – rosdistro rules data
- **rules_eval** – Function to apply to rule values, e.g. to convert variables. `fn(str)->str`

Returns `VcsConfig` subclass instance with interpreted rules data.

Example:

```
import rospkg

ros_root = rospkg.get_ros_root()

r = rospkg.RosPack()
depends = r.get_depends('roscpp')
path = r.get_path('rospy')
```

CHAPTER

SIX

COMMON API

exception `rospkg.`**`ResourceNotFound`**

Requested resource (e.g. package/stack) could not be found.

CHAPTER

SEVEN

INSTALLATION

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

```
pip install rospkg
```

or `easy_install`:

```
easy_install rospkg
```

CHAPTER

EIGHT

USING ROSPKG

The `rospkg` 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

NINE

ADVANCED: ROSPKG DEVELOPERS/CONTRIBUTORS

9.1 Developer's Guide

9.1.1 REP 114: rospkg standalone library

The rospkg library is being developed using the ROS REP process. It was introduced in REP 114: rospkg standalone library. Please read REP 114 to better understand the motivation and goals of the rospkg library.

9.1.2 Bug reports and feature requests

- Submit a bug report
- Submit a feature request

9.1.3 Developing new OsDetectors

Developing a new `OsDetector` is fairly straightforward. There are many examples in `os_detect.py`.

If you contribute a `OsDetector`, you *must* provide complete unit test coverage. For example, if your detector relies on parsing `/etc/issue` files, you must submit example `/etc/issue` files along with tests that parse them correctly.

Test files for os detection should be placed in `test/os_detect/os_name`.

If you submit a new detector, the documentation in `doc/os_detect.rst` must be updated as well.

9.1.4 Testing

Setup

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

rospkg uses Python nose for testing, which is a fairly simple and straightfoward test framework. You just have to write a function start with the name `test` and use normal `assert` statements for your tests.

rospkg also uses mock to create mocks for testing.

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

```
cd rospkg
nosetests test/*.py --with-coverage --cover-package=rospkg
```

9.1.5 Documentation

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

CHAPTER

TEN

INDICES AND TABLES

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

PYTHON MODULE INDEX

r