ROS Packages, ROS Overview

Summer Course on Developing on ROS Framework
Day 4

July 26, 2013

ROS Packages

Previously:

- `.launch` files
- `.msg` and `.srv` files

package.xml

Specifies package properties:

- Name
- Version
- Description
- Maintainer
- License
- Dependencies

Metapackage

A completely empty package that depends on other packages. Metapackages are a way to group packages.

CMakeLists.txt

- Build dependencies have to be repeated here
- Messages and services
- What to be passed to dependent packages
- Sources to be compiled and their dependencies
- Installation targets
- Unit tests

Package Structure

- `CMakeLists.txt` — CMake build file
- `manifest.xml` — Package Manifest
- `mainpage.dox` — Doxygen mainpage documentation
- `msg/` — Message (msg) types
- `srv/` — Service (srv) types
- `launch/` — Launch files
- `scripts/` — Executable scripts
- `include/package_name` — C++ include headers to export to other packages
- `src/` — C++ source files and headers that are not exported
- `src/package_name/` — Python source files that are exported to other packages

Package Structure

Store only what is needed in SVN or Git!

Generated directories like `bin/` or `msg_gen/` or backup files like those ending in ~ should never be committed. Use the `svn:ignore` property for SVN or the `.gitignore` file for Git to hide them and prevent mistakes.

Libraries

Generic nodes

Note that using generic nodes that read parameters and remapped topics is better than libraries! Use libraries only when this is not enough.

C++ provides a way to reuse code: Libraries. In ROS, to make some reusable generic functionality, it is common to create:

- A library package. This only generates a library file, with no executable node. This package should export the library file and the headers directory in the CMakeLists.txt.
- For each project/context that wants to use the library, create a package that depends on the library package. This uses the library in the context of the system.

Best Practices

- Keep packages small. A package should group things that make sense. If it gets too big, perhaps you should think about dividing it.
- Avoid committing big files. SVN and Git repositories can grow very fast. Use `catkin_download_test_data` or create your own script to download data from servers.

rosdep and rosinstall

- `rosdep` — Installs system dependencies
- `rosinstall` — Downloads ROS package sources automatically

Integrated Development Environments (IDEs)

- KDevelop
- QtCreator
- Eclipse CDT
- CodeBlocks
- NetBeans

Instructions for some:

`http://www.ros.org/wiki/IDEs`

Licensing with LGPL

- Add to every source code file:
  - Copyright notice (one line), such as:

    `Copyright 2012, 2013 Instituto de Sistemas e Robotica, Instituto Superior Tecnico`
  - Statement of copying permission (specifying the license)
- Include licenses in files `COPYING` and `COPYING.LESSER`

Full instructions

`http://www.gnu.org/licenses/gpl-howto.html`

ROS Overview

Simulation

- Stage — 2D Simulator
- Gazebo — 3D Simulator
- `/clock` and `/use_sim_time`
- Time and WallTime

Images

- Camera Drivers
  - Some ethernet cameras
  - Firewire cameras
  - USB cameras
  - Kinect
  - ...
- Image Pipeline
  - Calibration
  - Processing for monocular, stereo and depth images
  - Rotation
  - Visualization
- Nodelets

Tools: RQt

- Bag, Plot, Console, ...
- Create your own!

- RViz