Cygwin port of ROS (beta) – release notes

This document provides an overview of issues discovered while porting ROS to the Cygwin environment and provides general recommendation on building and using it.

If you encounter issues, let me know. gbrill@infusion.com

1. Building the port

To build the Cygwin ROS port, get a clean Cygwin installation, then:

1.1. Install the apt-cyg package and the tools it requires (wget, tar, gawk, xz and bzip2) along with their dependencies.
1.2. Create the /opt directory and ensure that your current user has write access to it
1.3. Copy the rosscripts directory to the /opt directory
1.4. Run **/opt/rosscripts/build_ros_isolated.sh**

Warning: some Cygwin mirrors may contain broken packages. If you encounter strange building errors, please ensure you are using the following Cygwin mirror:

http://lug.mtu.edu/

Note that the isolated build script resets the environment to avoid conflicts between Cygwin tools and other tools present in the PATH. If you modify the build scripts to depend on some of the environment variables, modify the **build_ros_isolated.sh** accordingly or call **build_ros.sh** instead.

The build script will use the following directories:

- The ROS tree will be located in **/opt/ros/install_isolated**
- Various 3rd-party libraries required by ROS that need to be built from sources will be built in **/opt/rosdeps** and installed to **/usr/local**

To rebuild a specific package run the **build_ros.sh** with the arguments supported by catkin_make, e.g. **build_ros.sh --pkg=<package name>**.

2. Using the port

Once a 64-bit port has been built, it is recommended to rebase all built libraries by closing ALL Cygwin instances and running the provided **rebase_ros.bat**.

Warning: If you are using a 32-bit Cygwin, doing a rebase can render the Cygwin installation unusable. Please make a full backup before running the rebase script.

Once it finishes running, open a Cygwin shell and run the following commands to get a working ROS environment:

```
/opt/ros/install_isolated/env.sh bash
export PATH=$PATH:/usr/local/lib:/opt/ros/install_isolated/lib
```

3. Resolved recurring issues

This section lists the most significant recurring issues found and resolved during the porting. In case of further build problems with the next ROS versions, please review the list below for troubleshooting clues.

3.1. Special Qt port

The Cygwin port of ROS includes a special port of the Qt framework that combines the Cygwin UNIX-like platform with the Win32 graphical subsystem (Q_WS_WIN32). This provides the following advantages:

- The port seamlessly integrates into the Cygwin environment and can be used by the UNIX-like code that does cannot be built using native Windows tools.
- The port provides native Windows GUI (as opposed to the slow X11-based GUI) with support for fast OpenGL.

3.2. Library directories

On Cygwin many of the ROS libraries are not installed into the **install_isolated/lib** directory and many others are installed into lib/<PACKAGE NAME> directory. This is caused by different CMake library path syntax on Windows and Linux (see http://www.cmake.org/Wiki/CMake:Install_Commands):

- On Linux the location of the .so file is specified using the **LIBRARY DESTINATION** statement. ROS packages use this statement to install the libraries into the lib directory.
- On Windows the location of the .dll file is specified using **RUNTIME DESTINATION** statement.
  - ROS packages that provide executable files use this statement globally to install the executable files into lib/<PACKAGE NAME>. As a side effect, this installs the Cygwin DLLs into lib/<PACKAGE NAME> as well. Unless each subdirectory of lib is added to PATH, Cygwin processes cannot locate those libraries.
  - ROS packages that do not provide executable files omit the **RUNTIME DESTINATION** statement. On Cygwin this prevents their DLLs from being installed.

The Cygwin port of ROS resolves this problem in 2 steps:

- The CATKIN_PACKAGE_BIN_DESTINATION variable is globally set to <ros>/bin instead of <ros>/lib/<package>. This has a side effect of placing all package executables to the <ros>/bin directory as well.
- Many packages that missed the **RUNTIME DESTINATION** statement now have it specified.

Potential problems:

- If you discover that some of the DLL files are not installed to <ros>/lib or <ros>/bin, locate the CMakeLists.txt file and add the RUNTIME DESTINATION statement explicitly, e.g.:

```
INSTALL(
        TARGETS ${PROJECT_NAME}
        ARCHIVE DESTINATION lib
        LIBRARY DESTINATION lib
      + RUNTIME DESTINATION lib
)
```

3.3. Python plugin names

Several ROS packages that provide Python plugins (e.g. TF) use the following syntax to specify the names for those plugins:

```
set_target_properties(pytf_py PROPERTIES OUTPUT_NAME tf PREFIX "_" SUFFIX ".so"
```

In this example the statement sets the library module name to **_tf.so**. This creates 2 problems:

- Hardcodes the .so extension that is not valid on Windows
- Creates a conflict between the import library for the normal TF dll and the Python DLL. Both import libraries are called **libtf.so** and the one that is built the last overrides the previously built one.

The Cygwin ROS port resolves this by changing the library name instead of the prefix:

```
set_target_properties(pytf_py PROPERTIES OUTPUT_NAME _tf PREFIX ""
```

3.4. Global name conflicts

Some of the ROS modules use names that conflict with preprocessor macros defined in Windows platform headers (e.g. NO_ERROR, DELETE, _B). This is solved by providing patches for the Windows platform headers.

4. Limitations

The following packages were excluded from the Cygwin port of ROS:

- gazebo_msgs
- gazebo_plugins
- gazebo_ros
- gazebo_ros_pkgs
- theora_image_transport

The 32-bit version of Cygwin has an implicit limit on the amount of DLLs that can be loaded at the same time. Hence various plugin-intensive applications like rqt may crash or report DLL remapping errors. If this occurs, please consider switching to a 64-bit build instead.