

Note: This tutorial assumes that you have completed the previous tutorials: Understanding ROS services and parameters (/ROS/Tutorials/UnderstandingServicesParams).

💡 Please ask about problems and questions regarding this tutorial on answers.ros.org (<http://answers.ros.org>). Don't forget to include in your question the link to this page, the versions of your OS & ROS, and also add appropriate tags.

Using rqt_console and roslaunch

Description: This tutorial introduces ROS using `rqt_console` (/rqt_console) and `rqt_logger_level` (/rqt_logger_level) for debugging and `roslaunch` (/roslaunch) for starting many nodes at once. If you use ROS fuerte or earlier distros where `rqt` (/rqt) isn't fully available, please see this page with this page (/ROS/Tutorials/UsingRxconsoleRoslaunch) that uses old rx based tools.

Tutorial Level: BEGINNER

Next Tutorial: Using rosed (/ROS/Tutorials/UsingRosEd)

Contents

1. Prerequisites rqt and turtlesim package
2. Using rqt_console and rqt_logger_level
 1. Quick Note about logger levels
 2. Using roslaunch
 3. The Launch File
 4. The Launch File Explained
 5. roslaunching

1. Prerequisites rqt and turtlesim package

The tutorial uses both the `rqt` and `turtlesim` packages. To do this tutorial, please install both packages, if you have not yet done so.

```
$ sudo apt-get install ros-<distro>-rqt ros-<distro>-rqt-common-plugins ros-<distro>-turtlesim
```

Replace `<distro>` with the name of your ROS distribution (/Distributions) (e.g. indigo, jade, kinetic, lunar...).

NOTE: you may have already built `rqt` and `turtlesim` for one of the previous tutorials. If you are not sure, installing them again will not hurt anything.

2. Using rqt_console and rqt_logger_level

`rqt_console` attaches to ROS's logging framework to display output from nodes. `rqt_logger_level` allows us to change the verbosity level (DEBUG, WARN, INFO, and ERROR) of nodes as they run.

Now let's look at the `turtlesim` output in `rqt_console` and switch logger levels in `rqt_logger_level` as we use `turtlesim`. Before we start the `turtlesim`, **in two new terminals** start `rqt_console` and `rqt_logger_level`:

```
$ rosrn rqt_console rqt_console
```

```
$ rosrn rqt_logger_level rqt_logger_level
```

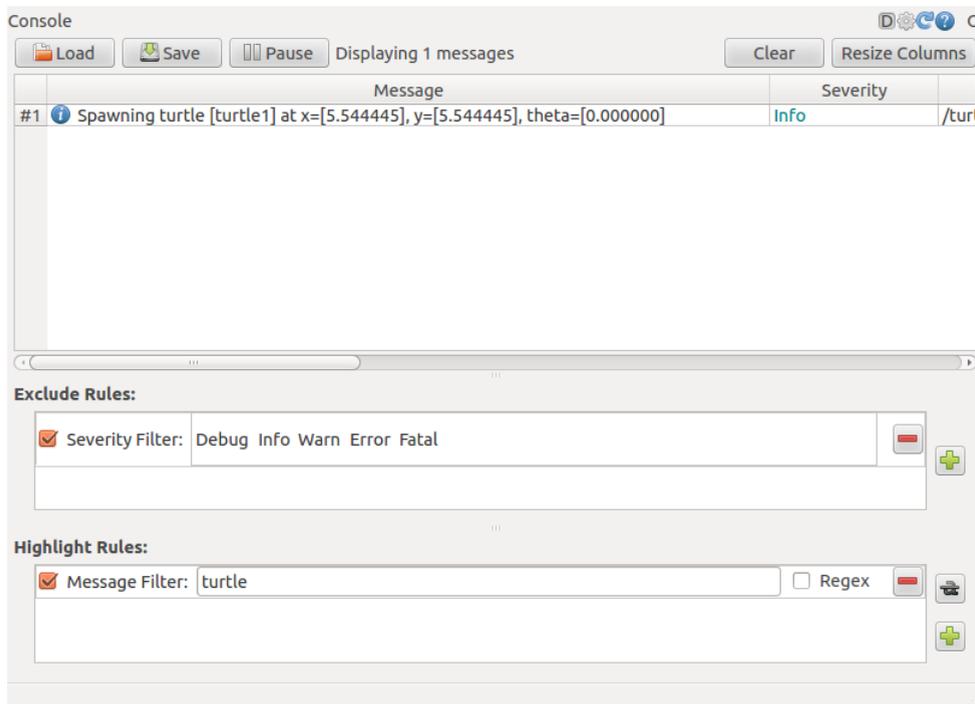
You will see two windows popup:

The image shows two windows from the Rqt GUI. The top window is the 'Console' window, which is currently empty and displays 'Displaying 0 messages'. It has buttons for 'Load', 'Save', 'Pause', 'Clear', and 'Resize Columns'. Below the message list are sections for 'Exclude Rules' and 'Highlight Rules'. The 'Exclude Rules' section has a checked 'Severity Filter' with a dropdown menu showing 'Debug Info Warn Error Fatal'. The 'Highlight Rules' section has a checked 'Message Filter' and an unchecked 'Regex' checkbox. The bottom window is the 'Logger Level' window, which has three columns: 'Nodes', 'Loggers', and 'Levels'. The 'Nodes' column lists '/rosout', '/rqt_gui_py_node_7714', and '/rqt_gui_py_node_7787', with '/rosout' selected. The 'Loggers' column lists 'ros', 'ros.roscpp', 'ros.roscpp.roscpp_internal', and 'ros.roscpp.superdebug'. The 'Levels' column lists 'Debug', 'Info', 'Warn', 'Error', and 'Fatal', with 'Info' selected. A 'Refresh' button is located at the bottom of the 'Nodes' column.

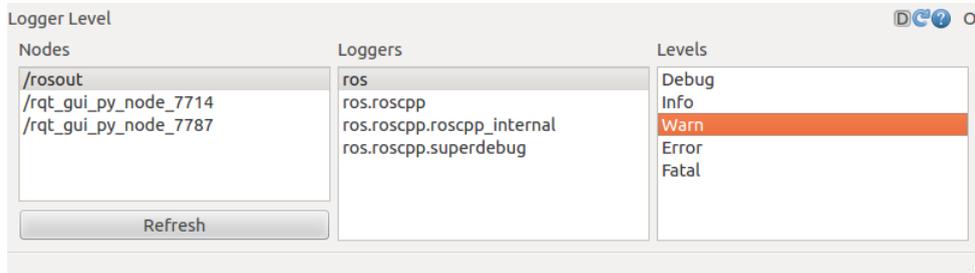
Now let's start turtlesim in a **new terminal**:

```
$ rosrn turtlesim turtlesim_node
```

Since the default logger level is INFO you will see any info that the turtlesim publishes when it starts up, which should look like:



Now let's change the logger level to Warn by refreshing the nodes in the `rqt_logger_level` window and selecting Warn as shown below:



Now let's run our turtle into the wall and see what is displayed in our `rqt_console`:

For ROS Hydro and later,

```
rostopic pub /turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '{linear: {x: 2.0, y: 0.0, z: 0.0}, angular: {x: 0.0,y: 0.0,z: 0.0}}'
```

For ROS Groovy and earlier,

```
rostopic pub /turtle1/command_velocity turtlesim/Velocity -r 1 -- 2.0 0.0
```

The screenshot shows the Rqt Console interface. At the top, there are buttons for 'Load', 'Save', 'Pause', 'Clear', and 'Resize Columns'. The main area displays a table of messages with columns for 'Message' and 'Severity'. The messages are all warnings with the text 'Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])'. Below the message list, there are sections for 'Exclude Rules' and 'Highlight Rules'. The 'Exclude Rules' section has a checked 'Severity Filter' with values 'Debug Info Warn Error Fatal'. The 'Highlight Rules' section has a checked 'Message Filter' with the value 'turtle' and an unchecked 'Regex' option.

#	Message	Severity
#141	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#140	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#139	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#138	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#137	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#136	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#135	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#134	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#133	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#132	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn
#131	Oh no! I hit the wall! (Clamping from [x=11.120889, y=5.544445])	Warn

Exclude Rules:

Severity Filter: Debug Info Warn Error Fatal

Highlight Rules:

Message Filter: turtle Regex

2.1 Quick Note about logger levels

Logging levels are prioritized in the following order:

```
Fatal
Error
Warn
Info
Debug
```

Fatal has the highest priority and Debug has the lowest. By setting the logger level, you will get all messages of that priority level or higher. For example, by setting the level to Warn, you will get all Warn, Error, and Fatal logging messages.

Let's Ctrl-C our turtlesim and let's use roslaunch to bring up multiple turtlesim nodes and a mimicking node to cause one turtlesim to mimic another:

2.2 Using roslaunch

roslaunch starts nodes as defined in a launch file.

Usage:

```
$ roslaunch [package] [filename.launch]
```

First go to the beginner_tutorials package we created (/ROS/Tutorials/CreatingPackage) and built (/ROS/Tutorials/BuildingPackages) earlier:

```
$ roscd beginner_tutorials
```

If roscd says something similar to *roscd: No such package/stack 'beginner_tutorials'*, you will need to source the environment setup file like you did at the end of the create_a_workspace (/catkin/Tutorials/create_a_workspace) tutorial:

```
$ cd ~/catkin_ws
$ source devel/setup.bash
$ roscd beginner_tutorials
```

Then let's make a launch directory:

```
$ mkdir launch
$ cd launch
```

NOTE: The directory to store launch files doesn't necessarily have to be named launch. In fact you don't even need to store them in a directory. roslaunch command automatically looks into the passed package and detects available launch files. However, this is considered good practice.

2.3 The Launch File

Now let's create a launch file called turtlemimic.launch and paste the following:

Toggle line numbers

```

1 <launch>
2
3 <group ns="turtlesim1">
4 <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
5 </group>
6
7 <group ns="turtlesim2">
8 <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
9 </group>
10
11 <node pkg="turtlesim" name="mimic" type="mimic">
12 <remap from="input" to="turtlesim1/turtle1"/>
13 <remap from="output" to="turtlesim2/turtle1"/>
14 </node>
15
16 </launch>

```

2.4 The Launch File Explained

Now, let's break the launch xml down.

Toggle line numbers

```
1 <launch>
```

Here we start the launch file with the launch tag, so that the file is identified as a launch file.

Toggle line numbers

```

3 <group ns="turtlesim1">
4 <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
5 </group>
6
7 <group ns="turtlesim2">
8 <node pkg="turtlesim" name="sim" type="turtlesim_node"/>
9 </group>

```

Here we start two groups with a namespace tag of turtlesim1 and turtlesim2 with a turtlesim node with a name of sim. This allows us to start two simulators without having name conflicts.

Toggle line numbers

```

11 <node pkg="turtlesim" name="mimic" type="mimic">
12 <remap from="input" to="turtlesim1/turtle1"/>
13 <remap from="output" to="turtlesim2/turtle1"/>
14 </node>

```

Here we start the mimic node with the topics input and output renamed to turtlesim1 and turtlesim2. This renaming will cause turtlesim2 to mimic turtlesim1.

Toggle line numbers

```
16 </launch>
```

This closes the xml tag for the launch file.

2.5 roslaunching

Now let's roslaunch the launch file:

```
$ roslaunch beginner_tutorials turtlemimic.launch
```

Two turtlesims will start and in a **new terminal** send the rostopic command:

For ROS Hydro and later,

```
$ rostopic pub /turtlesim1/turtle1/cmd_vel geometry_msgs/Twist -r 1 -- '[2.0, 0.0, 0.0]' '[0.0, 0.0, -1.8]'
```

For ROS Groovy and earlier,

```
$ rostopic pub /turtlesim1/turtle1/command_velocity turtlesim/Velocity -r 1 -- 2.0 -1.8
```

You will see the two turtlesims start moving even though the publish command is only being sent to turtlesim1.

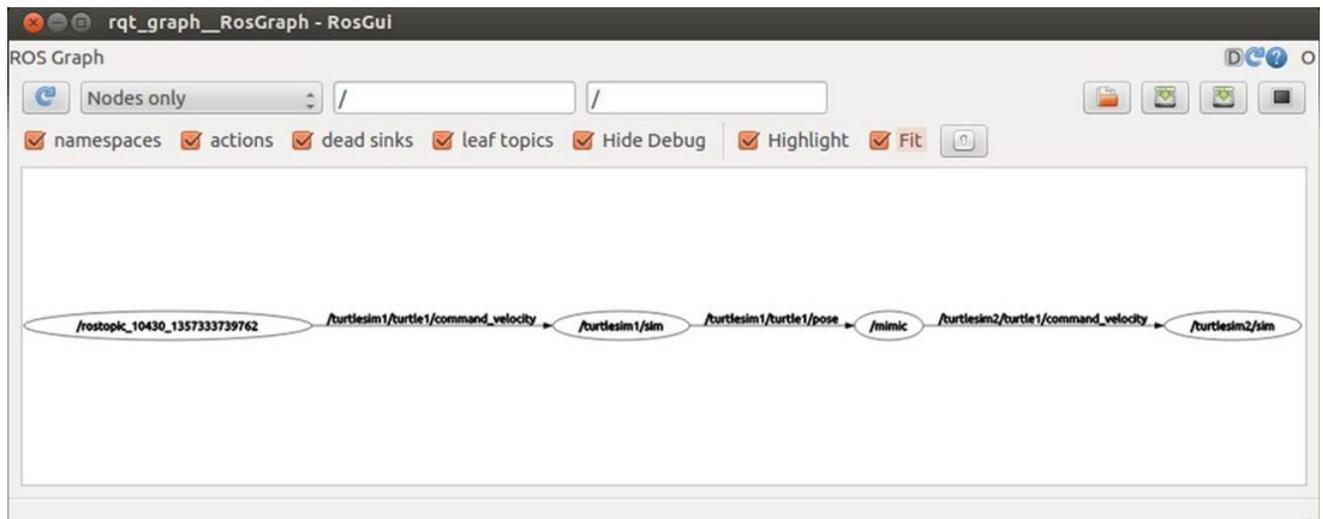


We can also use `rqt_graph` (`/rqt_graph`) to better understand what our launch file did. Run `rqt` (`/rqt`)'s main window and select *Plugins > Introspection > Node Graph*:

```
$ rqt
```

Or simply:

```
$ rqt_graph
```



Now that you have successfully used `rqt_console` and `roslaunch`, let's learn about editor options for ROS (`/ROS/Tutorials/UsingRosEd`). You can `Ctrl-C` all your turtlesims, as you will not need them for the next tutorials.

Except where otherwise noted, the ROS wiki is licensed under the Creative Commons Attribution 3.0 (<http://creativecommons.org/licenses/by/3.0/>) | Find us on Google+ (<https://plus.google.com/113789706402978299308>)

Wiki: `/ROS/Tutorials/UsingRqtconsoleRoslaunch` (last edited 2018-01-06 15:45:32 by ErikNewton (`/ErikNewton`))

Brought to you by: Open Source Robotics Foundation

(<http://www.osrfoundation.org>)