Understanding ROS Services and Parameters

**Description:** This tutorial introduces ROS services, and parameters as well as using the [rosservice](http://wiki.ros.org/rosservice) and [rosparam](http://wiki.ros.org/rosparam)commandline tools.  
  
**Tutorial Level:** BEGINNER  
  
**Next Tutorial:**  [Using rqt\_console and roslaunch](http://wiki.ros.org/ROS/Tutorials/UsingRqtconsoleRoslaunch) 

**Contents**

1. [ROS Services](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#ROS_Services)
2. [Using rosservice](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#Using_rosservice)
   1. [rosservice list](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#rosservice_list)
   2. [rosservice type](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#rosservice_type)
   3. [rosservice call](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#rosservice_call)
3. [Using rosparam](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#Using_rosparam)
   1. [rosparam list](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#rosparam_list)
   2. [rosparam set and rosparam get](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#rosparam_set_and_rosparam_get)
   3. [rosparam dump and rosparam load](http://wiki.ros.org/ROS/Tutorials/UnderstandingServicesParams#rosparam_dump_and_rosparam_load)

Assuming your turtlesim\_node is still running from the last tutorial, let's look at what services the turtlesim provides:

ROS Services

Services are another way that nodes can communicate with each other. Services allow nodes to send a **request** and receive a **response**.

Using rosservice

rosservice can easily attach to ROS's client/service framework with services. rosservice has many commands that can be used on services, as shown below:

Usage:

rosservice list print information about active services

rosservice call call the service with the provided args

rosservice type print service type

rosservice find find services by service type

rosservice uri print service ROSRPC uri

rosservice list

$ rosservice list

The list command shows us that the turtlesim node provides nine services: reset, clear, spawn, kill, turtle1/set\_pen, /turtle1/teleport\_absolute, /turtle1/teleport\_relative, turtlesim/get\_loggers, and turtlesim/set\_logger\_level. There are also two services related to the separate rosout node: /rosout/get\_loggers and /rosout/set\_logger\_level.

* /clear
* /kill
* /reset
* /rosout/get\_loggers
* /rosout/set\_logger\_level
* /spawn
* /teleop\_turtle/get\_loggers
* /teleop\_turtle/set\_logger\_level
* /turtle1/set\_pen
* /turtle1/teleport\_absolute
* /turtle1/teleport\_relative
* /turtlesim/get\_loggers
* /turtlesim/set\_logger\_level

Let's look more closely at the clear service using rosservice type:

rosservice type

Usage:

rosservice type [service]

Let's find out what type the clear service is:

$ rosservice type /clear

* std\_srvs/Empty

This service is empty, this means when the service call is made it takes no arguments (i.e. it sends no data when making a **request** and receives no data when receiving a **response**). Let's call this service using rosservice call:

rosservice call

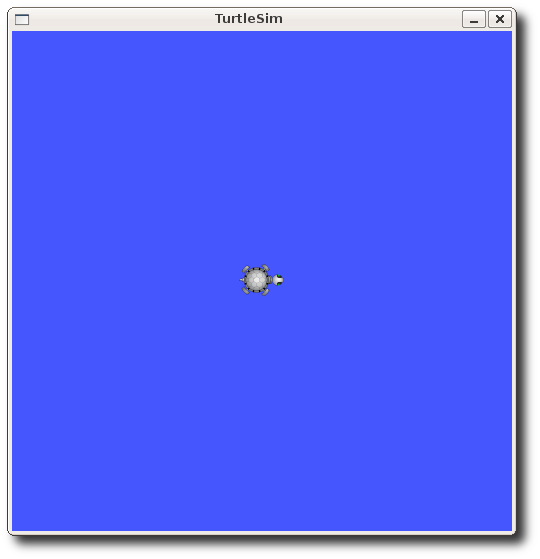
Usage:

rosservice call [service] [args]

Here we'll call with no arguments because the service is of type empty:

$ rosservice call /clear

This does what we expect, it clears the background of the turtlesim\_node.

* 

Let's look at the case where the service has arguments by looking at the information for the service spawn:

$ rosservice type /spawn | rossrv show

* float32 x
* float32 y
* float32 theta
* string name
* ---
* string name

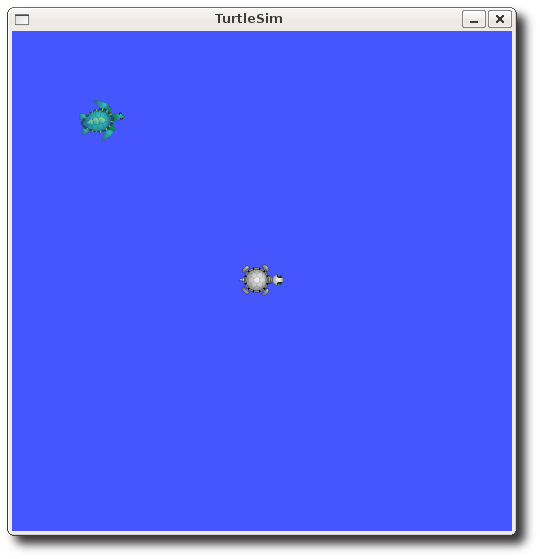
This service lets us spawn a new turtle at a given location and orientation. The name field is optional, so let's not give our new turtle a name and let turtlesim create one for us.

$ rosservice call /spawn 2 2 0.2 ""

The service call returns with the name of the newly created turtle

* name: turtle2

Now our turtlesim should look like this:

* 

Using rosparam

rosparam allows you to store and manipulate data on the ROS [Parameter Server](http://wiki.ros.org/Parameter%20Server). The Parameter Server can store integers, floats, boolean, dictionaries, and lists. rosparam uses the YAML markup language for syntax. In simple cases, YAML looks very natural: 1 is an integer, 1.0 is a float, one is a string, true is a boolean, [1, 2, 3] is a list of integers, and {a: b, c: d} is a dictionary. rosparam has many commands that can be used on parameters, as shown below:

Usage:

rosparam set set parameter

rosparam get get parameter

rosparam load load parameters from file

rosparam dump dump parameters to file

rosparam delete delete parameter

rosparam list list parameter names

Let's look at what parameters are currently on the param server:

rosparam list

$ rosparam list

Here we can see that the turtlesim node has three parameters on the param server for background color:

* /background\_b
* /background\_g
* /background\_r
* /rosdistro
* /roslaunch/uris/host\_57aea0986fef\_\_34309
* /rosversion
* /run\_id

Let's change one of the parameter values using rosparam set:

rosparam set and rosparam get

Usage:

rosparam set [param\_name]

rosparam get [param\_name]

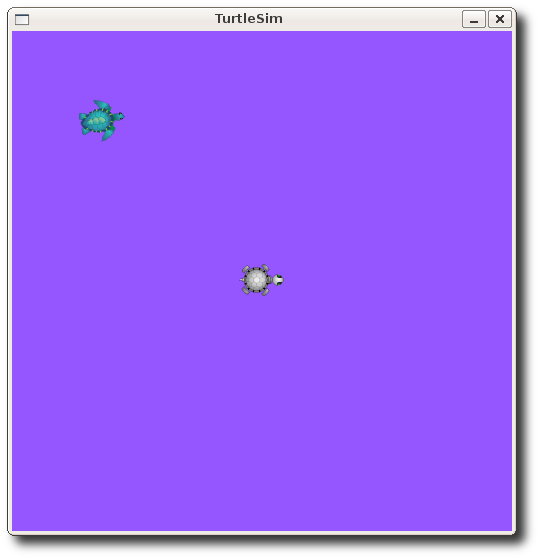
Here will change the red channel of the background color:

$ rosparam set /background\_r 150

This changes the parameter value, now we have to call the clear service for the parameter change to take effect:

$ rosservice call /clear

Now our turtlesim looks like this:

* 

Now let's look at the values of other parameters on the param server. Let's get the value of the green background channel:

$ rosparam get /background\_g

* 86

We can also use rosparam get / to show us the contents of the entire Parameter Server.

$ rosparam get /

* background\_b: 255
* background\_g: 86
* background\_r: 150
* roslaunch:
* uris: {'aqy:51932': 'http://aqy:51932/'}
* run\_id: e07ea71e-98df-11de-8875-001b21201aa8

You may wish to store this in a file so that you can reload it at another time. This is easy using rosparam:

rosparam dump and rosparam load

Usage:

rosparam dump [file\_name] [namespace]

rosparam load [file\_name] [namespace]

Here we write all the parameters to the file params.yaml

$ rosparam dump params.yaml

You can even load these yaml files into new namespaces, e.g. copy:

$ rosparam load params.yaml copy

$ rosparam get /copy/background\_b

* 255