Note: This tutorial assumes that you have completed the previous tutorials: Understanding Topics (/ROS/Tutorials/UnderstandingTopics), Understanding ServicesParams (/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 Parameters in roscpp

Description: This tutorial will show you the `NodeHandle` parameter API, allowing you to manipulate parameters from the Parameter Server (/Parameter%20Server).

Tutorial Level: BEGINNER

Next Tutorial: Accessing Private Names with NodeHandle (/roscpp_tutorials/Tutorials/AccessingPrivateNamesWithNodeHandle)

Contents

1. Retrieving Parameters

There are two methods to retrieve parameters with `NodeHandle`. In the following example code, `n` is an instance of `NodeHandle`.

1.1 getParam()

`getParam()` has a number of overloads which all follow the same basic form:

Toggle line numbers

```
1 bool getParam (const std::string& key, parameter_type& output_value) cons
t
```

- **key** is a Graph Resource Name (/Names)
- **output_value** is the place to put the retrieved data, where parameter_type is one of bool, int, double, string, or a special `XmlRpcValue` type which can represent any type and also lists/maps.

Use of `getParam()` is fairly simple:

Toggle line numbers

```
1     std::string s;
2     n.getParam("my_param", s);
```

Note that `getParam()` returns a bool, which provides the ability to check if retrieving the parameter succeeded or not:

Toggle line numbers

```
std::string s;
if (n.getParam("my_param", s))
{
  ROS_INFO("Got param: %s", s.c_str());
}
else
{
  ROS_ERROR("Failed to get param 'my_param'");
}
```

1.2 param()

`param()` is similar to `getParam()`, but allows you to specify a default value in the case that the parameter could not be retrieved:

Toggle line numbers

```
1   int i;
2   n.param("my_num", i, 42);
```

Sometimes the compiler requires a hint for the string type.

Toggle line numbers

```
1   std::string s;
2   n.param<std::string>("my_param", s, "default_value");
```

2. Setting Parameters

Setting parameters is done through the `setParam()` methods:

Toggle line numbers

```
1   n.setParam("my_param", "hello there");
```

`setParam()`, like `getParam()`, can take bool, int, double, string, and a special `XmlRpcValue` type

3. Deleting Parameters

Deleting parameters is done through the `deleteParam()` method:

Toggle line numbers

```
1   n.deleteParam("my_param");
```

4. Checking for Existence

This is not usually necessary, but there is a `hasParam()` method that allows you to check for a parameter's existence:

Toggle line numbers

```
1   if (!n.hasParam("my_param"))
2   {
3     ROS_INFO("No param named 'my_param'");
4   }
```

5. Searching for Parameters

The Parameter Server (/Parameter%20Server) allows you to "search" for parameters, starting at your namespace and working through your parent namespaces.

For example, if the parameter `/a/b` exists in the parameter server, and your `NodeHandle` is in the `/a/c` namespace, `searchParam()` for `b` will yield `/a/b`. However, if parameter `/a/c/b` is added, `searchParam()` for `b` will now yield `/a/c/b`.

Toggle line numbers

```
1   std::string param_name;
2   if (n.searchParam("b", param_name))
3   {
4     // Found parameter, can now query it using param_name
5     int i = 0;
6     n.getParam(param_name, i);
7   }
8   else
9   {
10    ROS_INFO("No param 'b' found in an upward search");
11  }
```

Next Tutorial: Accessing Private Names with NodeHandle (/roscpp_tutorials/Tutorials/AccessingPrivateNamesWithNodeHandle)

Wiki: roscpp_tutorials/Tutorials/Parameters (last edited 2018-05-08 07:43:27 by AndreaPonza (/AndreaPonza))

(http://www.osrfoundation.org)