roscpp overview (/roscpp/Overview): *Initialization and Shutdown (/roscpp/Overview/Initialization%20and%20Shutdown) | Basics (/roscpp/Overview/Messages) | Advanced: Traits [ROS C Turtle] (/roscpp/Overview/MessagesTraits) | Advanced: Custom Allocators [ROS C Turtle] (/roscpp/Overview/MessagesCustomAllocators) | Advanced: Serialization and Adapting Types [ROS C Turtle] (/roscpp/Overview/MessagesSerializationAndAdaptingTypes) | Publishers and Subscribers (/roscpp/Overview/Publishers%20and%20Subscribers) | Services (/roscpp/Overview/Services) | Parameter Server | Timers (Periodic Callbacks) (/roscpp/Overview/Timers) | NodeHandles (/roscpp/Overview/NodeHandles) | Callbacks and Spinning (/roscpp/Overview/Callbacks%20and%20Spinning) | Logging (/roscpp/Overview/Logging) | Names and Node Information (/roscpp/Overview/Names%20and%20Node%20Information) | Time (/roscpp/Overview/Time) | Exceptions (/roscpp/Overview/Exceptions) | Compilation Options (/roscpp/Overview/Compilation%20Options) | Advanced: Internals (/roscpp/Overview/Internals) | tf/Overview (/tf/Overview) | tf/Tutorials (/tf/Tutorials) | C++ Style Guide (/CppStyleGuide)*

Contents

The ROS Parameter Server (/Parameter%20Server) can store strings, integers, floats, booleans, lists, dictionaries, iso8601 dates, and base64-encoded data. Dictionaries must have string keys.

roscpp (/roscpp)'s parameter API supports all of these, though it is only easy to use strings, integers, floats and booleans. Support for the other options is done using the XmlRpc::XmlRpcValue class (http://docs.ros.org/api/xmlrpcpp/html/classXmlRpc_1_1XmlRpcValue.html)

roscpp (/roscpp) has two different parameter APIs: the "bare" versions which live in the `ros::param` namespace, and the "handle" versions which are called through the `ros::NodeHandle` interface. Both versions will be explained for each operation below.

1. Getting Parameters

Fetch a value from the parameter server. Each version supports strings, integers, doubles, booleans and XmlRpc::XmlRpcValue (http://docs.ros.org/api/xmlrpcpp/html/classXmlRpc_1_1XmlRpcValue.html)s. `false` is returned if the parameter does not exist, or is not of the right type. There is also a version that returns a default value.

`ros::NodeHandle::getParam()`

Parameters retrieved through the `NodeHandle` version are resolved relative to that `NodeHandle`'s namespace. See the roscpp NodeHandles overview (/roscpp/Overview/NodeHandles) for more information.

Toggle line numbers

```
ros::NodeHandle nh;
std::string global_name, relative_name, default_param;
if (nh.getParam("/global_name", global_name))
{
  ...
}

if (nh.getParam("relative_name", relative_name))
{
...
}

// Default value version
nh.param<std::string>("default_param", default_param,
"default_value");
```

`ros::param::get()`

Parameters retrieved through the "bare" version are resolved relative to the node's namespace.

Toggle line numbers

```
std::string global_name, relative_name, default_param;
if (ros::param::get("/global_name", global_name))
{
  ...
}

if (ros::param::get("relative_name", relative_name))
{
...
}

// Default value version
ros::param::param<std::string>("default_param", default_param, "defa
ult_value");
```

1.1 Cached Parameters

`ros::NodeHandle::getParamCached()` and `ros::param::getCached()` provide local caching of parameter data. Using these versions informs the Parameter Server (/Parameter%20Server) that this node would like to be notified when the parameter is changed, and prevents the node from having to re-lookup the value with the parameter server on subsequent calls.

Cached parameters are a significant speed increase (after the first call), but should be used sparingly to avoid overloading the master. Cached parameters are also currently less reliable in the case of intermittent connection problems between your node and the master.

2. Setting Parameters

Similar to getting parameters, each version supports strings, integers, doubles, booleans and XmlRpc::XmlRpcValue (http://docs.ros.org/api/xmlrpcpp/html/classXmlRpc_1_1XmlRpcValue.html)s.

`ros::NodeHandle::setParam()`

Parameters retrieved through the `NodeHandle` version are resolved relative to that `NodeHandle`'s namespace. See the roscpp NodeHandle overview (/roscpp/Overview/NodeHandles) for more information.

Toggle line numbers

```
ros::NodeHandle nh;
nh.setParam("/global_param", 5);
nh.setParam("relative_param", "my_string");
nh.setParam("bool_param", false);
```

`ros::param::set()`

Parameters retrieved through the "bare" version are resolved relative to the node's namespace.

Toggle line numbers

```
ros::param::set("/global_param", 5);
ros::param::set("relative_param", "my_string");
ros::param::set("bool_param", false);
```

3. Checking Parameter Existence

`ros::NodeHandle::hasParam()`

Toggle line numbers

```
ros::NodeHandle nh;
if (nh.hasParam("my_param"))
{
  ...
}
```

`ros::param::has()`

Toggle line numbers

```
if (ros::param::has("my_param"))
{
  ...
}
```

4. Deleting Parameters

`ros::NodeHandle::deleteParam()`

Toggle line numbers

```
ros::NodeHandle nh;
nh.deleteParam("my_param");
```

`ros::param::del()`

Toggle line numbers

```
ros::param::del("my_param");
```

5. Accessing Private Parameters

Accessing private parameters is done differently depending on whether you're using the "handle" or "bare" interfaces. In the handle interface you must create a new `ros::NodeHandle` with the private namespace as its namespace:

Toggle line numbers

```
ros::NodeHandle nh("~");
std::string param;
nh.getParam("private_name", param);
```

In the bare interface you can access private parameters with the same notation used to describe them, e.g.:

Toggle line numbers

```
std::string param;
ros::param::get("~private_name", param);
```

6. Searching for Parameter Keys

There are times where you want to get a parameter from the *closest* namespace. For example, if you have a "`robot_name`" parameter, you just want to search upwards from your private namespace until you find a matching parameter. Similarly, if you have a group of camera nodes, you may wish to set some parameters commonly in a shared namespace but override others by setting them in a private (`~name`) namespace.

Note: in order to use `search` effectively, you should use it with *relative* names instead of `/global` and `~private` names.

`ros::NodeHandle::searchParam()`

Toggle line numbers

```
std::string key;
if (nh.searchParam("bar", key))
{
  std::string val;
  nh.getParam(key, val);
}
```

`ros::param::search()`

Toggle line numbers

```
std::string key;
if (ros::param::search("bar", key))
{
  std::string val;
  ros::param::get(key, val);
}
```

7. Retrieving Lists

New in ROS groovy

You can get and set lists and dictionaries of primitives and strings as `std::vector` and `std::map` containers with the following templated value types:

- bool
- int
- float
- double
- string

For example, you can get vectors and maps with both the `ros::NodeHandle::getParam` / `ros::NodeHandle::setParam` interface or the `ros::param::get` / `ros::param::set` interface:

Toggle line numbers

```
// Create a ROS node handle
ros::NodeHandle nh;

// Construct a map of strings
std::map<std::string,std::string> map_s, map_s2;
map_s["a"] = "foo";
map_s["b"] = "bar";
map_s["c"] = "baz";

// Set and get a map of strings
nh.setParam("my_string_map", map_s);
nh.getParam("my_string_map", map_s2);

// Sum a list of doubles from the parameter server
std::vector<double> my_double_list;
double sum = 0;
nh.getParam("my_double_list", my_double_list);
for(unsigned i=0; i < my_double_list.size(); i++) {
  sum += my_double_list[i];
}
```

On ROS Fuerte and earlier, lists on the parameter server can only be retreived through the use of the XmlRpc::XmlRpcValue (http://docs.ros.org/api/xmlrpcpp/html/classXmlRpc_1_1XmlRpcValue.html) class, which can represent any of the types on the parameter server. This is still a valid method in later ROS versions.

Toggle line numbers

```
XmlRpc::XmlRpcValue my_list;
nh.getParam("my_list", my_list);
ROS_ASSERT(my_list.getType() == XmlRpc::XmlRpcValue::TypeArray);

for (int32_t i = 0; i < my_list.size(); ++i)
{
  ROS_ASSERT(my_list[i].getType() == XmlRpc::XmlRpcValue::TypeDouble);
  sum += static_cast<double>(my_list[i]);
}
```