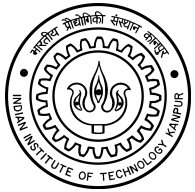


ROS

ROBOTICS CLUB

SCIENCE AND TECHNOLOGY COUNCIL
IIT KANPUR

Robotics
Club IITKanpur

Contents

- *Using ROS Parameters*
- *Writing custom ROS messages*
- *Using ROS Services*

Robotics Club IITKanpur

The Parameter Server

The Parameter server is a part of the ROS Network. It saves/provides the values of specific variables (parameters) for the nodes of that package to use during runtime.

Parameters are best used to set values of static configuration parameters (therefore saved in the `config` folder of the package)

The Parameter Server can store integers, floats, boolean, dictionaries, and lists.

These can be specified in a *launch* file or, also as separate *parameter* files.

Writing *parameter* files

The parameter files are written in the YAML format.

For example:

```
camera:
  left:
    name: left_camera
    exposure: 1
  right:
    name: right_camera
    exposure: 1.1
```

This specifies the parameter values for the left and right cameras respectively.

This is saved as
camera/left/name = ‘left_camera’
camera/left/exposure = 1
camera/right/name = ‘right_camera’
camera/right/exposure = 1.1

YAML: **Y**AML **A**in’t **M**arkup **L**anguage

Robotics Club IITKanpur

Adding *parameters* to *launch* files

<rosparam file=”$(find [PACKAGE_NAME])/config/params.yaml” />

- Must be inside the <launch> </launch> block
- If you put it inside a <node> </node> block, params will be loaded under the node’s **namespace**

Namespace:
For example, /turtle1 was the turtlesim node’s namespace.
/ is the Global/Master namespace.

Robotics Club IITKanpur

Adding ROS Parameters

- First, create a launch file to launch both your talker and listener nodes.
- Now create the **config** directory and create **params.yaml** in it.
- We will specify the topic name and publish rate as parameters

topic: speak
rate: 10

This goes in params.yaml

- Then include the param file into your launch file

Robotics Club IITKanpur

Adding parameters to Nodes

- Parameters are published on server by launch file.
- We need to now access them inside the node.
- NodeHandle has functions to fetch parameters from server

nh.getParam(“[param_name]”, var_name);

OR

nh.param(“[param_name]”, var_name, def_value);

Parameters - Publisher Node

```
#include <ros/ros.h>
#include <std_msgs/String.h>

int main(int argc, char** argv){
        ros::init(argc, argv, “talker”);
        ros::NodeHandle nh;
        std::string topic; double rate;
        nh.getParam(“topic”, topic);
        nh.getParam(“rate”, rate);
        ros::Publisher pub = nh.advertise<std_msgs::String>(topic,10);
        ros::Rate loopRate(rate);
        int count = 0;
        while(ros::ok()){
                std_msgs::String msg;
                msg.data = “Hello World ” + std::to_string(count) ;
                pub.publish(msg);
                loopRate.sleep();
                count++;}
return 0;}
```

Specify the proper path for the parameters.
If the path is not correct, the param won’t get added

Robotics Club IITKanpur

Parameters - Subscriber Node

```
#include <ros/ros.h>
#include <std_msgs/String.h>

std::string data;
void subCallback(std_msgs::String msg){
        data = msg.data;
        ROS_INFO("%s", data.c_str());
return;}

int main(int argc, char** argv){
        ros::init(argc, argv, "listener");
        ros::NodeHandle nh;
        std::string topic_name; double rate_val;
        nh.getParam("topic", topic_name);
        nh.getParam("rate", rate_val);

        // continued at the right
```

```
// continued from left

        ros::Subscriber sub =
nh.subscribe(topic_name, 10, subCallback);
        ros::Rate loopRate(rate_val);

        while(ros::ok()){
                ros::spinOnce();
                loopRate.sleep();
        }

return 0;}
```

Robotics Club IITKanpur

Creating custom messages

- Messages are basically structs
- Can define and use our own structs in the **msg** folder
- Message files must end with **.msg**
- Need to do two more things:

1. Edit package.xml: Add **message_generation** and **message_runtime** as dependencies
2. Edit CMakeLists.txt

Creating custom messages

- Let's create a custom message **Data**, for our talker-listener package. Its definition:

```
std_msgs/String message
int32 a
int32 b
```


This goes in a file named **Data.msg** inside the **msg** folder of the package.

Case-sensitive, space-sensitive.

Robotics Club IITKanpur

Creating custom messages
(editing *CMakeLists.txt*)

1. find_package()

```
7   ## Find catkin macros and libraries
8   ## if COMPONENTS list like find_package(catkin REQUIRED COMPONENTS xyz)
9   ## is used, also find other catkin packages
10  find_package(catkin REQUIRED COMPONENTS
11    roscpp
12    std_msgs
13  )
```

Before

```
10  find_package(catkin REQUIRED COMPONENTS
11    roscpp
12    std_msgs
13    message_generation
14  )      You, 12 hours ago • Session 2
```

After

Robotics Club IITKanpur

Creating custom messages
(editing *CMakeLists.txt*)

2. add_message_files()

```
49  ## Generate messages in the 'msg' folder
50  # add_message_files(
51  #   FILES
52  #   Message1.msg
53  #   Message2.msg
54  # )
```

Before

```
49  ## Generate messages in the 'msg' folder
50  add_message_files(
51    FILES
52    Data.msg
53  #   Message1.msg
54  #   Message2.msg
55  )
```

After

Robotics Club IITKanpur

Creating custom messages
(editing *CMakeLists.txt*)

3. generate_messages()

```
71  ## Generate added messages and services with any dependencies listed here
72  # generate_messages(
73  #   DEPENDENCIES      You, 12 hours ago • Session 2
74  #   std_msgs  # Or other packages containing msgs
75  # )
```

Before

```
71  ## Generate added messages and services with any depe
72  generate_messages(
73    DEPENDENCIES
74    std_msgs  # Or other packages containing msgs
75  )
```

After

Robotics Club IITKanpur

Creating custom messages
(editing *CMakeLists.txt*)

4. catkin_package()

Before

```
106  catkin_package(
107  #  INCLUDE_DIRS include
108  #  LIBRARIES talker-listener
109  #  CATKIN_DEPENDS roscpp
110  #  DEPENDS system_lib
111  )
```

After

```
106  catkin_package(
107  #  INCLUDE_DIRS include
108  #  LIBRARIES talker-listener
109     CATKIN_DEPENDS roscpp message_runtime
110  #  DEPENDS system_lib
111  )
```

Robotics Club IITKanpur

Renaming your package

- Build error:

  ERROR: package name 'talker-listener' is illegal and cannot be used in message generation.

- Fix:
  - Change second line of CMakeLists.txt (project())
  - Change <name> in package.xml
  - Rename folder (use **mv**)
    - Not necessary but good practice to keep package name same as package folder name
- **Clean** and then build again

ROS Naming Conventions

- First character is an alpha character ([a-z|A-Z]) or tilde (~)
- Subsequent characters can be alphanumeric ([0-9|a-z|A-Z]), underscores (_)

This is why we got the previous error: ‘-’ is not allowed.

For more, refer: https://wiki.ros.org/Names

Robotics Club IITKanpur

Using custom messages

- Very similar to using the existing messages
- Your custom message resides within your package
- Message header to include **Data.msg** from **talker_listener** package:

```
#include <talker_listener/Data.h>
```

- To create a message object,

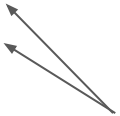
```
talker_listener::Data msg;
```

Robotics Club IITKanpur

Publishing custom messages

```
#include <ros/ros.h>
#include <talker_listener/Data.h>

int main(int argc, char** argv){
        ros::init(argc, argv, “talker”);
        ros::NodeHandle nh; std::string topic; double rate;
        nh.getParam(“topic”, topic); nh.getParam(“rate”, rate);
        ros::Publisher pub = nh.advertise<talker_listener::Data>(topic,10);
        ros::Rate loopRate(rate);
        int count1 = 0, count2 = 1, count3 = 2;
        while(ros::ok()){
                talker_listener::Data msg;
                msg.message.data = “Hello World ” + std::to_string(count1) ;
                msg.a = count2; msg.b = count3;
                pub.publish(msg);
                loopRate.sleep();
                count1++; count2+=2; count3+=2;}
return 0;}
```



Exactly according to message definition

Robotics Club IITKanpur

Subscribing to custom messages

```
#include <ros/ros.h>
#include <talker_listener/Data.h>

std::string data; int a, b;
void subCallback(talker_listener::Data msg){
        data = msg.message.data;
        a = msg.a; b = msg.b;
        ROS_INFO("%s %d %d", data.c_str(), a, b);
return;}

int main(int argc, char** argv){
        ros::init(argc, argv, "listener");
        ros::NodeHandle nh;
        std::string topic; double rate;
        nh.getParam("topic", topic);
        nh.getParam("rate", rate);
```

```
// continued from left

        ros::Subscriber sub = nh.subscribe(topic, 10,
subCallback);
        ros::Rate loopRate(rate);

        while(ros::ok()){
                ros::spinOnce();
                loopRate.sleep();
        }

return 0;}
```

Robotics Club IITKanpur

ROS Services

- Services behave like topics.
- But nothing is constantly published.
- Returns a **response** only when a **request** is given
- **std_srvs** provides a few basic services.
- Can implement custom services like messages (out of scope for now)
- ROS provides two objects:
    - **ServiceServer:** advertises the service
    - **ServiceClient:** calls an existing service

Creating a Service Server

- Let's use the **std_srvs/Trigger** service to stop our talker.
- For this, you will need to create a **ServiceServer** in the talker

```
ros::ServiceServer server = nh.advertiseService("[service_name]", [CALLBACK_FUNC]);
```

- The actual work is done inside the callback

```
bool callback(std_srvs::Trigger::Request& req, std_srvs::Trigger::Response& resp){

        // do stuff here using req data
        // put stuff inside resp

        return true;
}
```

Robotics Club IITKanpur

Adding the Service Server

```
#include <ros/ros.h>
#include <talker_listener/Data.h>
#include <std_srvs/Trigger.h>

bool flag = true;
bool serverCallback(std_srvs::Trigger::Request &req, std_srvs::Trigger::Response &resp){
        flag = !flag;
        resp.success = true;
        resp.message = “Triggered”;
        return true;}
…
        ros::Publisher pub = nh.advertise<talker_listener::Data>(topic,10);
        ros::ServiceServer server = nh.advertiseService(“pause”, serverCallback);
        while(ros::ok()){
                ...
                if(flag) pub.publish(msg);
                ros::spinOnce();
        ...
```

First add **std_srvs** as a dependency

Note that req is empty for this service

Robotics Club IITKanpur

Using Service Clients

- Service Clients are used to call existing services within a node

```
ros::ServiceClient client = nh.serviceClient<[SERVICE_TYPE]>("[service_name]")
```

- Let's create a new node that calls the publisher's service after a fixed interval of time, say **5 seconds**.

Adding Service Clients

Remember to add executable

```
#include <ros/ros.h>
#include <std_srvs/Trigger.h>

int main(int argc, char** argv){
        ros::init(argc, argv, “trigger”);
        ros::NodeHandle nh;
        ros::ServiceClient client = nh.serviceClient(“pause”);
        ros::Rate loopRate(30);
        std_srvs::Trigger srv;
        int count = 0;
        while(ros::ok()){
                ros::spinOnce();
                if(count % 150 == 0) client.call(srv);        // called every 150/30 seconds.
                loopRate.sleep(); count++;
        }
return 0;}
```

Since our service has an empty request, we only create a service object. Otherwise, you would have to set the values of the request fields.

Robotics Club IITKanpur

Contact us if you have any problem/suggestion:

Abhay Varshney	**9559015388**
Madhur Deep Jain	**8894051687**
Neil Shirude	**9850892135**

roboticsclubiitkanpur@gmail.com
http://students.iitk.ac.in/roboclub/
https://www.facebook.com/roboclubiitkanpur
https://www.youtube.com/c/RoboticsClubIITKanpur

Robotics Club IITKanpur