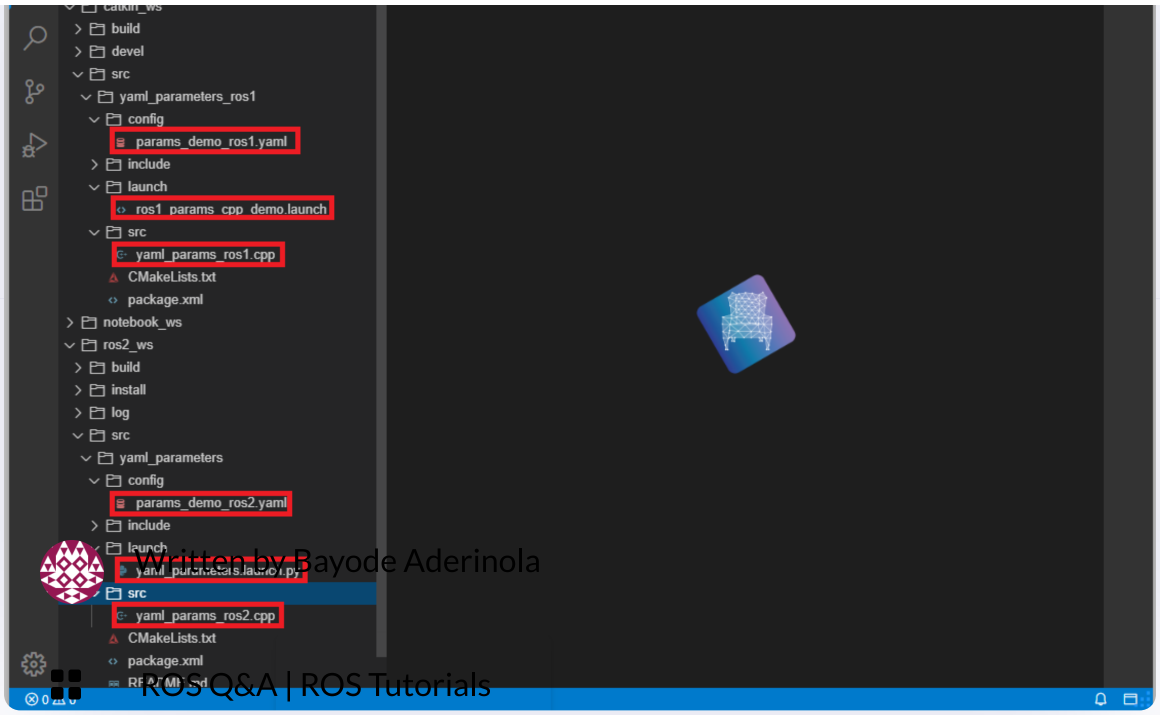
How to read and write parameters in ros1 and ros2



```
catkin_ws
  build
  devel
  src
    yaml_parameters_ros1
      config
        params_demo_ros1.yaml
      include
      launch
        ros1_params_cpp_demo.launch
      src
        yaml_params_ros1.cpp
      CMakeLists.txt
      package.xml
notebook_ws
ros2_ws
  build
  install
  log
  src
    yaml_parameters
      config
        params_demo_ros2.yaml
      include
      launch
        yaml_parameters.launch.py
      src
        yaml_params_ros2.cpp
      CMakeLists.txt
      package.xml
      README.md
```

Written by Bayode Aderinola

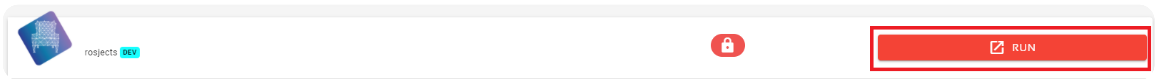
ROS Q&A | ROS Tutorials

13/07/2022

In this post, you will learn how to read and write parameters in ros1 and ros2, using C++ nodes. You will see the slight differences in the ros1 and ros2 nodes and parameter files.

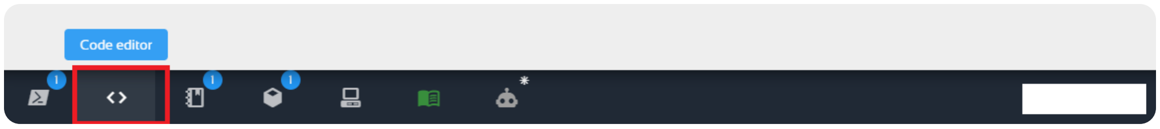
Step 1: Get a Copy of the ROS package containing the code used in the post

Click here to copy the project. It would be copied to your cloud account at The Construct. That done, open the project using the ***Run*** button. This might take a few moments, please be patient.

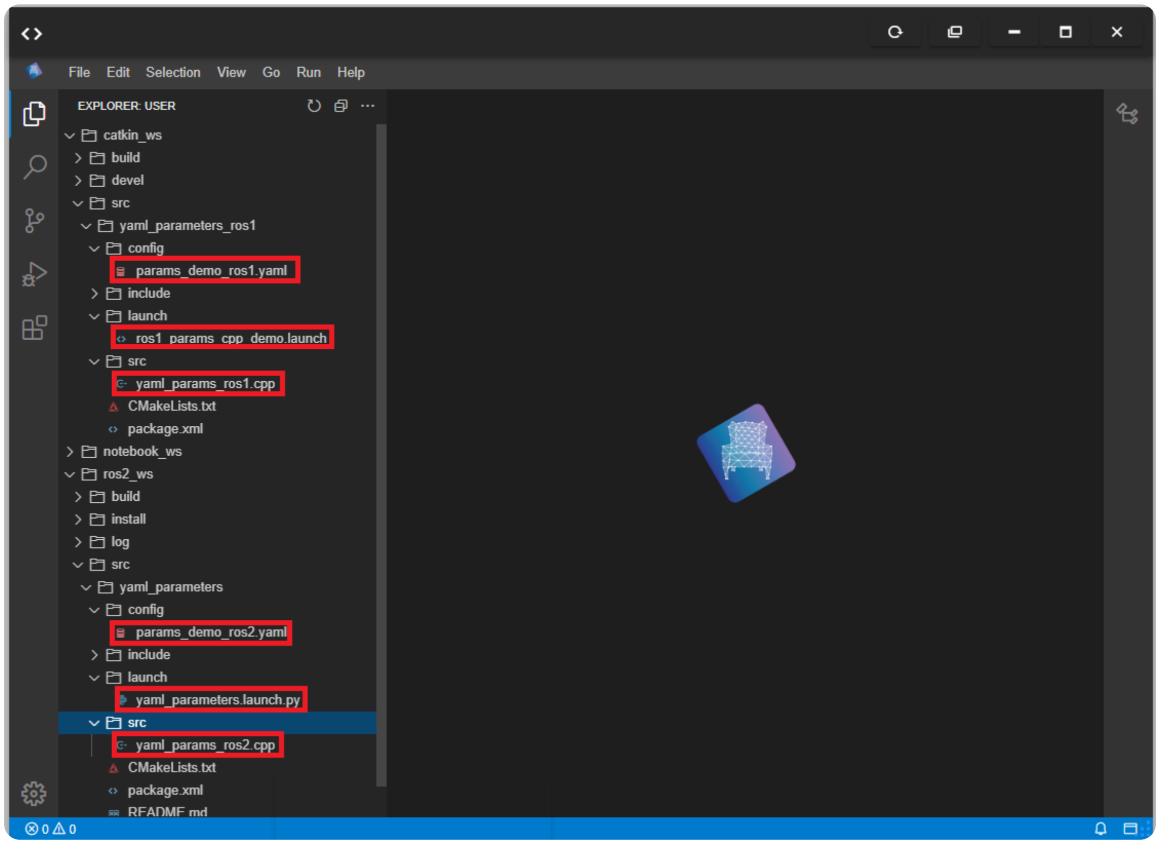


PS: If you don’t have an account on the The Construct, you would need to create one. Once you create an account or log in, you will be able to follow the steps to case you need to read on and duplicate the source code of the package in your own local workspace. However, please note that we cannot support local PCs and you will have to fix any errors you run into on your own.

Step 2: Explore the source code using the IDE



Open the IDE by clicking on the icon as shown above. You should now see something similar to the image below:



The six main files we will work with in this post are highlighted in red in the image above. These files are:

ROS1:

2. `catkin_ws/src/yaml_parameters_ros1/launch/ros1_params_cpp_demo.launch`
3. `catkin_ws/src/yaml_parameters_ros1/src/yaml_params_ros1.cpp`

ROS2:

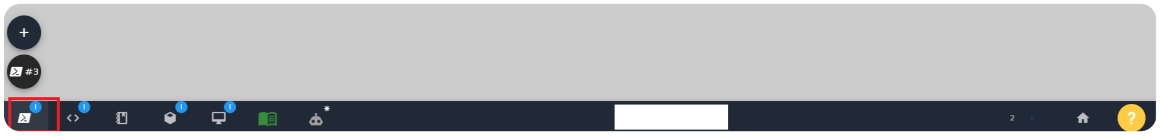
1. `ros2_ws/src/yaml_parameters/config/params_demo_ros2.yaml`
2. `ros2_ws/src/yaml_parameters/launch/yaml_parameters.launch.py`
3. `ros2_ws/src/yaml_parameters/src/yaml_params_ros2.cpp`

Double-click on each of the files in the IDE to open and study the contents. We will examine some of these files in the following steps.

Step 3: Understand how to read and write (load) parameters in ROS1

Now it's time to see how to read and write parameters in ros1, working in the ros1 workspace.

Open a web shell and run the following commands:



```
cd ~/catkin_ws
source /opt/ros/noetic/setup.bash
source devel/setup.bash
roscore
```

The code block above changes to the ros1 workspace, sources it, and then starts the `roscore` (needed for ros1). Now let's see a list of the current ros1 parameters available. Open another web shell and type the following:

```
/rosdistro
/roslaunch/uris/host_1_xterm__41731
/rosversion
/run_id
```

On the same web shell, run the following command to print out the ros parameters:

```
rosrun yaml_parameters_ros1 yaml_parameters_ros1_node
```

Your output should be the following:

```
[ INFO] [1657670307.494093666]: Integer parameter: 1
[ INFO] [1657670307.495689754]: Double parameter: 0.100000
[ INFO] [1657670307.495741256]: String parameter: default
[ INFO] [1657670307.495773876]: Nested integer parameter: 1
[ INFO] [1657670307.495797588]: Nested string parameter: default
[ INFO] [1657670307.495819891]: Boolean parameter: 0
```

The logic that produced the output above in contained in the `catkin_ws/src/yaml_parameters_ros1/src/yaml_params_ros1.cpp` file. Let's see its content.

```cpp
#include "ros/ros.h"
#include <string>

int main(int argc, char **argv) {
  ros::init(argc, argv, "my_node");

  ros::NodeHandle nh;

  int param0 = 1;
  double param1 = 0.1;
  std::string param2 = "default";
  int p3weight = 1;
  std::string p3name = "default";
  bool param4 = false;

  nh.getParam("param0", param0);
  nh.getParam("param1", param1);
  nh.getParam("param2", param2);
  nh.getParam("param3/weight", p3weight);
  nh.getParam("param3/name", p3name);
  nh.getParam("param4", param4);
  nh.getParam("param5", param5);
  nh.getParam("param6", param6);
  nh.getParam("param7", param7);
  nh.getParam("param8", param8);

  ROS_INFO("Integer parameter: %d", param0);
  ROS_INFO("Double parameter: %f", param1);
  ROS_INFO("String parameter: %s", param2.c_str());
  ROS_INFO("Nested integer parameter: %d", p3weight);
  ROS_INFO("Nested string parameter: %s", p3name.c_str());
  ROS_INFO("Boolean parameter: %d", param4);
  ROS_INFO("Boolean vector parameter [0]: %d", static_cast<int>(param5[0]));
  ROS_INFO("Integer vector parameter [0]: %d", static_cast<int>(param6[0]));
  ROS_INFO("Double vector parameter [0]: %f", static_cast<double>(param7[0]));
  ROS_INFO("String vector parameter [0]: %s", param8[0].c_str());

  return 0;
}
```

But wait…are we getting the parameters in the YAML file (`catkin_ws/src/yaml_parameters_ros1/config/params_demo_ros1.yaml`) and their correct values? Let's see what's the in there!

```
# interger array
param0: 2
# double
param1: 0.2
# string
param2: "R2-D2"
# nested parameters
param3:
  weight: 2
  name: "wood"
```

```
# double array
param7: [0.2, 0.3, 0.4, 0.5]
# string array
param8: ["Bedroom", "Bathroom", "Laundry room",
"Kitchen", "Living room"]
```

Gosh, we are not getting these parameters nor their values, and you probably know why! So far we have been reading the parameters but have loaded them. Now let's get that done: enter the launch file `catkin_ws/src/yaml_parameters_ros1/launch/ros1_params_cpp_demo.launch`.

```
<launch>
    <rosparam file="$(find
yaml_parameters_ros1)/config/params_demo_ros1.yaml"
/>
</launch>
```

This file, when launched, loads the YAML parameter file. Let's see that in action.

Run the following command in the open web shell:

```
roslaunch yaml_parameters_ros1
ros1_params_cpp_demo.launch
```

You should get something like the following:

```
... logging to /home/user/.ros/log/c5f51462-023c-
11ed-bb5c-0242ac180007/roslaunch-1_xterm-13868.log
Checking log directory for disk usage. This may take
a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://1_xterm:45437/

SUMMARY
========

PARAMETERS
 * /param0: 2
 * /param1: 0.2
 * /param7: [0.2, 0.3, 0.4, 0.5]
 * /param8: ['Bedroom', 'Bath...
 * /rosdistro: noetic
 * /rosversion: 1.15.11

NODES

ROS_MASTER_URI=http://1_xterm:11311

No processes to monitor
shutting down processing monitor...
... shutting down processing monitor complete
```

Well we have some fancy output there, but what has changed? Let's see that by running two previous commands:

```
rosparam list
rosrun yaml_parameters_ros1 yaml_parameters_ros1_node
```

Your output should now look like this:

```
user:~/catkin_ws$ rosparam list
/param0
/param1
/param2
/param3/name
/param3/weight
/param4
/param5
/param6
/param7
/param8
/rosdistro
/roslaunch/uris/host_1_xterm__41731
/roslaunch/uris/host_1_xterm__45437
/rosversion
/run_id
user:~/catkin_ws$ rosrun yaml_parameters_ros1
yaml_parameters_ros1_node
[ INFO] [1657671513.558912623]: Integer parameter: 2
[ INFO] [1657671513.560352415]: Double parameter:
0.200000
[ INFO] [1657671513.560386238]: String parameter: R2-
D2
```

```
parameter [0]: 1
[ INFO] [1657671513.560465819]: Integer vector
parameter [0]: 5
[ INFO] [1657671513.560484622]: Double vector
parameter [0]: 0.200000
[ INFO] [1657671513.560497372]: String vector
parameter [0]: Bedroom
```

Can you spot the differences between the formal and the latter outputs of these commands? Sure you can! So, well, that's how to read and load parameter in ros1!

Step 4: Understand how to read and write (load) parameters in ROS2

Now let's change to the ros2 workspace.

```
cd ~/ros2_ws
source /opt/ros/foxy/setup.bash
source install/setup.bash
```

In ros2 we need to have a node running before we can check for parameters, because there is no parameter server in ros2. Let's try running the node then. The logic behind this node is contained in the `ros2_ws/src/yaml_parameters/src/yaml_params_ros2.cpp` file:

```
#include <rclcpp/rclcpp.hpp>

class MainNode : public rclcpp::Node {
public:
  MainNode() : rclcpp::Node("node",
rclcpp::NodeOptions()) {

    // example: declare parameters, default value
given
    declare_parameter("param0", 1);
    declare_parameter("param1", 0.1);
    declare_parameter("param2", "default");
    declare_parameter("param3.weight", 1);
    declare_parameter("param3.name", "default");
    declare_parameter("param4", false);
    // example: declare a variable when declaring a
    declare_parameter("param7", std::vector<double>(4, 0.1));
    declare_parameter("param8", std::vector<std::string>(5, "default"));

    // Get parameter values one by one
    auto p0 = get_parameter("param0").as_int();
    auto p1 = get_parameter("param1").as_double();
    auto p2 = get_parameter("param2").as_string();
    auto p3weight = get_parameter("param3.weight").as_int();
    auto p3name = get_parameter("param3.name").as_string();
    auto p4 = get_parameter("param4").as_bool();
    auto p5 = get_parameter("param5").as_bool_array();
    auto p6 = get_parameter("param6").as_integer_array();
    auto p7 = get_parameter("param7").as_double_array();
    auto p8 = get_parameter("param8").as_string_array();

    // Print parameters
    RCLCPP_INFO(get_logger(), "Integer parameter: %ld", p0);
    RCLCPP_INFO(get_logger(), "Double parameter: %f", p1);
    RCLCPP_INFO(get_logger(), "String parameter: %s", p2.c_str());
    RCLCPP_INFO(get_logger(), "Nested integer parameter: %ld", p3weight);
    RCLCPP_INFO(get_logger(), "Nested string parameter: %s", p3name.c_str());
    RCLCPP_INFO(get_logger(), "Boolean parameter: %d", p4);
    RCLCPP_INFO(get_logger(), "Boolean vector parameter [0]: %d",
                static_cast<int>(p5[0]));
    RCLCPP_INFO(get_logger(), "Integer vector parameter [0]: %d",
                static_cast<int>(p6[0]));
    RCLCPP_INFO(get_logger(), "Double vector parameter [0]: %f",
                static_cast<double>(p7[0]));
    RCLCPP_INFO(get_logger(), "String vector parameter [0]: %s", p8[0].c_str());
  }
};
```

```
  rclcpp::shutdown();
  return 0;
}
```

Go for it: run the node:

```
ros2 run yaml_parameters main_node
```

The output will be something like:

```
INFO] [1657672243.344585940] [node]: Integer parameter: 1
[INFO] [1657672243.344674910] [node]: Double parameter: 0.100000
[INFO] [1657672243.344704157] [node]: String parameter: default
[INFO] [1657672243.344720646] [node]: Nested integer parameter: 1
[INFO] [1657672243.344730700] [node]: Nested string parameter: default
[INFO] [1657672243.344745410] [node]: Boolean parameter: 0
[INFO] [1657672243.344755485] [node]: Boolean vector parameter [0]: 0
[INFO] [1657672243.344769947] [node]: Integer vector parameter [0]: 1
[INFO] [1657672243.344780509] [node]: Double vector parameter [0]: 0.100000
[INFO] [1657672243.344795534] [node]: String vector parameter [0]: default
```

Next, let's get the list of ROS2 parameters:

```
user:~$ ros2 param list
/node:
  param0
  param1
  param2
  param3.name
  param3.weight
  param4
  param5
  param6
```

`ros2_ws/src/yaml_parameters/config/params_demo_ros2.yaml`?

```
# if a namespace is specified
# ns_name:
# node name
parameter_types_example:
  ros__parameters:
    # int
    param0: 2
    # double
    param1: 0.2
    # string
    param2: "R2-D2"
    # nested parameters
    param3:
      weight: 2
      name: "wood"
    # boolean
    param4: true
    # boolean array
    param5: [true, true, true]
    # interger array
    param6: [5,6,7,8]
    # double array
    param7: [0.2, 0.3, 0.4, 0.5]
    # string array
    param8: ["Bedroom", "Bathroom", "Laundry room",
"Kitchen", "Living room"]
```

No, we are not :(. But not to worry, the launch file `ros2_ws/src/yaml_parameters/launch/yaml_parameters.launch.py` comes to the rescue! Let's examine its content.

```
#!/usr/bin/env python3

import os
from launch import LaunchDescription
from launch_ros.actions import Node
from ament_index_python.packages import
get_package_share_directory


def generate_launch_description():
```

```
                get_package_share_directory('yaml_parameters')
                 'config', 'params_demo_ros2.yaml')],
            output='screen'),
    ])
```

Oh my, it's a Python! Let's set it loose and see what happens! Stop the currently running program with Ctrl + C and run the following in its place and check that your output is similar.

```
user:~/ros2_ws$ ros2 launch yaml_parameters yaml_parameters.launch.py
[INFO] [launch]: All log files can be found below /home/user/.ros/log/2022-07-13-00-40-35-558545-1_xterm-18432
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [main_node-1]: process started with pid [18434]
[main_node-1] [INFO] [1657672835.736702673] [parameter_types_example]: Integer parameter: 2
[main_node-1] [INFO] [1657672835.736792219] [parameter_types_example]: Double parameter: 0.200000
[main_node-1] [INFO] [1657672835.736810397] [parameter_types_example]: String parameter: R2-D2
[main_node-1] [INFO] [1657672835.736842588] [parameter_types_example]: Nested integer parameter: 2
[main_node-1] [INFO] [1657672835.736847188] [parameter_types_example]: Nested string parameter: wood
[main_node-1] [INFO] [1657672835.736855303] [parameter_types_example]: Boolean parameter: 1
[main_node-1] [INFO] [1657672835.736863129] [parameter_types_example]: Boolean vector parameter [0]: 1
[main_node-1] [INFO] [1657672835.736870819] [parameter_types_example]: Integer vector parameter [0]: 5
[main_node-1] [INFO] [1657672835.736878422] [parameter_types_example]: Double vector parameter [0]: 0.200000
[main_node-1] [INFO] [1657672835.736887246] [parameter_types_example]: String vector parameter [0]: Bedroom
```