



LAB #4: ROS2 USING RCLPY IN JULIA

Samar trabelsi
Dept. of EE
ISET Bizerte — Tunisia
 *Satrabelsi*

Nehed Ouhibi
Dept. of EE
ISET Bizerte — Tunisia
 *N1ehed*

we are required to carry out this lab using the REPL as in Figure 1.

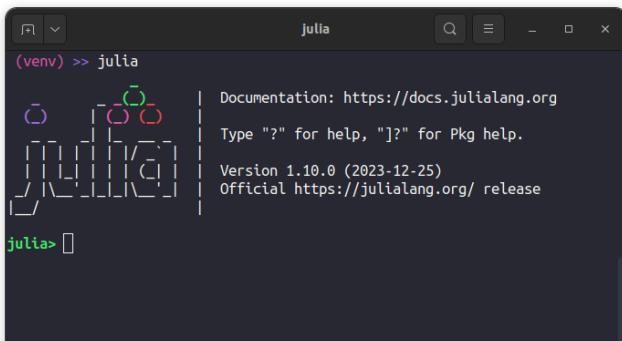


Figure 1: Julia REPL

Exo 1: Minimal Publisher/Subscriber Setup

The combination of Julia and rclpy opens up opportunities for developing efficient and performant robotics applications with the benefits of ROS2s ecosystem.

Make sure to read the instructions thoroughly, follow each step precisely, and ask for clarification if needed. We begin first of all by sourcing our ROS2 installation as follows:

```
source /opt/ros/humble/setup.zsh
```

Always start by sourcing ROS2 installation in any newly opened terminal.

Open a *tmux* session and write the instructions provided at your Julia REPL.

```
using PyCall
# Import the rclpy module from ROS2 Python
rclpy = pyimport("rclpy")
str = pyimport("std_msgs.msg")

# Initialize ROS2 runtime
rclpy.init()
```

```
# Create node
node = rclpy.create_node("my_publisher")
rclpy.spin_once(node, timeout_sec=1)

# Create a publisher, specify the message type and
the topic name
pub = node.create_publisher(str.String,
"infodev", 10)

# Publish the message `txt`
for i in range(1, 100)
    msg = str.String(data="Hello, ROS2 from Julia!"
    $(string(i)))
    pub.publish(msg)
    txt = "[TALKER] " * msg.data
    @info txt
    sleep(1)
end

# Cleanup
rclpy.shutdown()
node.destroy_node()
```

This is a Julia script that uses the PyCall package to interface with ROS2 (Robot Operating System 2) via its Python API. Here's a breakdown of what the code does:

1. Imports: The

```
pyimport
```

function is used to import Python modules into Julia. Here, it's importing

```
rclpy
```

(the main ROS2 Python module) and

```
std_msgs.msg
```

(a standard message type in ROS2).

2. Initialize ROS2:

```
rclpy.init()
```

initializes the ROS2 runtime system.

3. Create Node:

```
rclpy.create_node("my_publisher")
```

creates a new ROS2 node named "my_publisher". A node is an entity that processes data.

```
rclpy.spin_once(node, timeout_sec=1)
```

allows the node to process its callbacks once.

4. Create Publisher:

```
node.create_publisher(str.String,  
"infodev", 10)
```

creates a publisher that can send messages of type

```
std_msgs.msg.String
```

on the "infodev" topic. The

```
10
```

is the queue size for outgoing messages.

5. Publish Messages: The

```
for
```

loop creates a new

```
std_msgs.msg.String
```

message, publishes it on the "infodev" topic, logs the message data with the prefix "[TALKER]", and then waits for 1 second. This is done 100 times.

6. Cleanup: Finally,

```
rclpy.shutdown()
```

shuts down the ROS2 runtime system and

```
node.destroy_node()
```

destroys the node.

In summary, this script creates a ROS2 node that publishes a series of string messages ("Hello, ROS2 from Julia! (i)") to the "infodev" topic. It's a basic example of a publisher node in ROS2.

In a newly opened terminal, we need to setup a subscriber that listens to the messages being broadcasted by our previous publisher¹.

```
using PyCall

rclpy = pyimport("rclpy")
str = pyimport("std_msgs.msg")

# Initialization
rclpy.init()

# Create node
node = rclpy.create_node("my_subscriber")

# Callback function to process received messages
function callback(msg)
    txt = "[LISTENER] I heard: " * msg.data
    @info txt
end

# Create a ROS2 subscription
sub = node.create_subscription(str.String,
"infodev", callback, 10)

while rclpy.ok()
    rclpy.spin_once(node)
end

# Cleanup
node.destroy_node()
rclpy.shutdown()
```

This is another Julia script that uses the PyCall package to interface with ROS2 (Robot Operating System 2) via its Python API. However, unlike the previous script, this one creates a subscriber node in ROS2. Here's a breakdown of what the code does:

1. Imports: The

```
pyimport
```

function is used to import Python modules into Julia. Here, it's importing

```
rclpy
```

(the main ROS2 Python module) and

```
std_msgs.msg
```

(a standard message type in ROS2).

2. Initialize ROS2:

```
rclpy.init()
```

¹Remember to source ROS2 installation before using it with Julia

initializes the ROS2 runtime system.

3. Create Node:

```
rclpy.create_node("my_subscriber")
```

creates a new ROS2 node named "my_subscriber". A node is an entity that processes data.

4. Callback Function: The

```
callback
```

function is defined to process incoming messages. When a message is received, it logs the message data with the prefix "[LISTENER] I heard: ".

5. Create Subscription:

```
node.create_subscription(str.String,  
"infodev", callback, 10)
```

creates a subscription that listens for messages of type

```
std_msgs.msg.String
```

on the "infodev" topic. The

```
callback
```

function is called for each incoming message. The

```
10
```

is the queue size for incoming messages.

6. Spin: The

```
while
```

loop keeps the node running and processing its callbacks (i.e., the

```
callback
```

function) as long as ROS2 is still running (

```
rclpy.ok()
```

returns

```
true
```

).

7. Cleanup: Finally,

```
node.destroy_node()
```

destroys the node and

```
rclpy.shutdown()
```

shuts down the ROS2 runtime system.

In summary, this script creates a ROS2 node that listens for string messages on the "infodev" topic and logs each message it receives.

The graphical tool **rqt_graph** of Figure 2 displays the flow of data between our nodes: *my_publisher* and *my_subscriber*, through the topic we designed *infodev*.

```
source /opt/ros/humble/setup.zsh  
rqt_graph
```

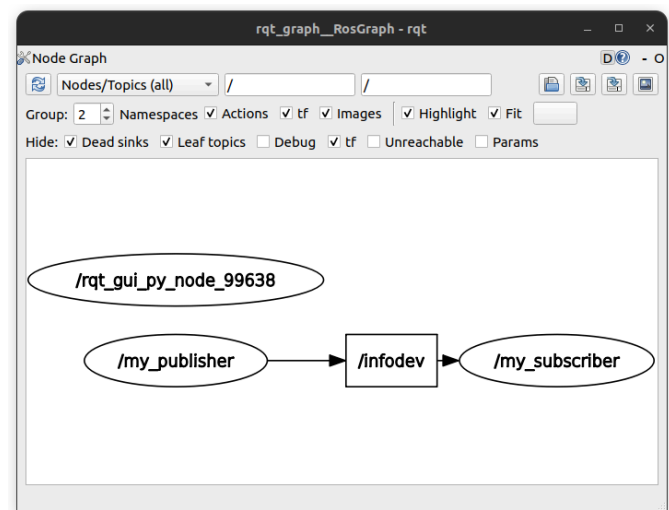
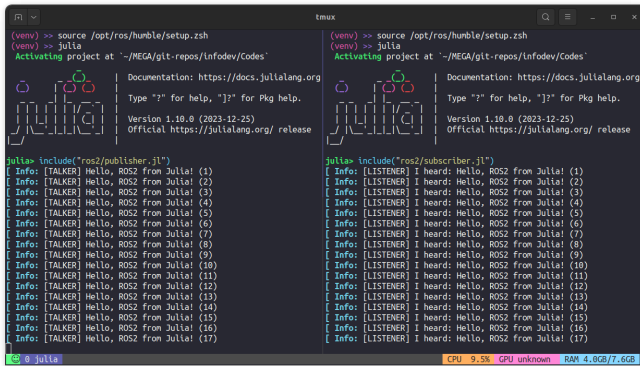


Figure 2: rqt_graph

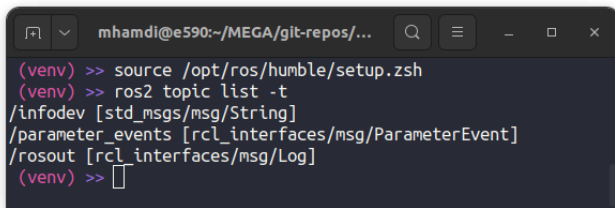
Figure 3 depicts the publication and reception of the message "Hello, ROS2 from Julia!" in a terminal. The left part of the terminal showcases the message being published, while the right part demonstrates how the message is being received and heard.



The image shows two terminal windows side-by-side. The left window is a ROS2 publisher terminal where the user has sourced the environment and runned `ros2 run infodev codes`. It shows 17 messages being published: `[Info: [TALKER] Hello, ROS2 from Julia! (1)]` through `(17)`. The right window is a ROS2 subscriber terminal where the user has sourced the environment and runned `ros2 run infodev codes`. It shows 17 messages being received: `[Info: [LISTENER] I heard: Hello, ROS2 from Julia! (1)]` through `(17)`. Both windows show the ROS2 logo and version information (1.18.0 (2023-12-25)).

Figure 3: Minimal publisher/subscriber in ROS2

Figure 4 shows the current active topics, along with their corresponding interfaces.



The image shows a terminal window with the command `ros2 topic list -t` executed. The output lists three active topics and their interfaces: `/infodev [std_msgs/msg/String]`, `/parameter_events [rc_l_interfaces/msg/ParameterEvent]`, and `/rosout [rc_l_interfaces/msg/Log]`. The terminal window title is `mhamdi@e590:~/MEGA/git-repos/...`.

Figure 4: List of topics