Example #2: Action Client - ROS-MQTT Bridge

Aim

- To write a ROS Node which will act as Action Client.
- Create this Action Client Node in `pkg_ros_iot_bridge` ROS Package.
- The name of the action use by this Action Client should be `/action_ros_iot`.
- The Action Client should send goals to perform pub-sub MQTT tasks to the Action Server discussed in the previous example.
- A custom action message file which should be called `msgIotRos.action` should be used with the following content,

```
# goal
string protocol
string mode
string topic
string message
---
# result
bool flag_success
---
# feedback
int8 percentage_complete
```

- The Action Client should first send a Goal to the Server to publish on a MQTT Topic by sending the following things in the Goal message.
  - protocol = `mqtt`
  - mode = `pub`
  - topic = `/eyrc/<unique_id>/ros_to_iot`
  - message = `Hello from ROS!`
- After that the Action Client should send a Goal to the server to subscribe to a MQTT Topic by sending the following things in the Goal message.
  - protocol = `mqtt`
  - mode = `sub`
  - topic = `/eyrc/<unique_id>/iot_to_ros`
  - message = `NA`
- The Action Client should also be subscribed to the ROS Topic mentioned in `sub_cb_ros_topic` field of `config_ros_iot.yaml` (discussed in the previous example) and it should listen for incoming messages from this ROS Topic Subscription.

Code

`node_simple_action_server_turtle.py`

```python
#!/usr/bin/env python

# ROS Node - Action Client - IoT ROS Bridge

import rospy
import actionlib

from pkg_iot_ros_bridge.msg import msgIotRosAction      # Message Class that is used by F
from pkg_iot_ros_bridge.msg import msgIotRosGoal        # Message Class that is used for
from pkg_iot_ros_bridge.msg import msgIotRosResult      # Message Class that is used for

class IotRosBridgeActionClient:

    # Constructor
    def __init__(self):

        # Initialize Action Client
        self._ac = actionlib.ActionClient('/action_iot_ros',
                                          msgIotRosAction)

        # Dictionary to Store all the goal handels
        self._goal_handles = {}

        # Store the MQTT Topic on which to Publish in a variable
        param_config_iot = rospy.get_param('config_iot')
        self._config_mqtt_pub_topic = param_config_iot['mqtt']['topic_pub']

        # Wait for Action Server that will use the action - '/action_iot_ros' to start
        self._ac.wait_for_server()
        rospy.loginfo("Action server up, we can send goals.")


    # This function will be called when there is a change of state in the Action Client S
    def on_transition(self, goal_handle):

        # from on_goal() to on_transition(). goal_handle generated by send_goal() is used

        result = msgIotRosResult()

        index = 0
        for i in self._goal_handles:
            if self._goal_handles[i] == goal_handle:
                index = i
                break

        rospy.loginfo("Transition Callback. Client Goal Handle #: " + str(index))
        rospy.loginfo("Comm. State: " + str(goal_handle.get_comm_state()) )
        rospy.loginfo("Goal Status: " + str(goal_handle.get_goal_status()) )

        # Comm State - Monitors the State Machine of the Client which is different from S
        # Comm State = 2 -> Active
        # Comm State = 3 -> Wating for Result
        # Comm State = 7 -> Done

        # if (Comm State == ACTIVE)
        if goal_handle.get_comm_state() == 2:
            rospy.loginfo(str(index) + ": Goal just went active.")

        # if (Comm State == DONE)
        if goal_handle.get_comm_state() == 7:
            rospy.loginfo(str(index) + ": Goal is DONE")
            rospy.loginfo(goal_handle.get_terminal_state())

            # get_result() gets the result produced by the Action Server
            result = goal_handle.get_result()
            rospy.loginfo(result.flag_success)

            if (result.flag_success == True):
                rospy.loginfo("Goal successfully completed. Client Goal Handle #: " + str
            else:
                rospy.loginfo("Goal failed. Client Goal Handle #: " + str(index))


    # This function is used to send Goals to Action Server
    def send_goal(self, arg_protocol, arg_mode, arg_topic, arg_message):
        # Create a Goal Message object
        goal = msgIotRosGoal()

        goal.protocol = arg_protocol
        goal.mode = arg_mode
        goal.topic = arg_topic
        goal.message = arg_message
```

```
        rospy.loginfo("Send goal.")

        # self.on_transition - It is a function pointer to a function which will be calle
        #                      there is a change of state in the Action Client State Mac
        goal_handle = self._ac.send_goal(goal,
                                         self.on_transition,
                                         None)

        return goal_handle

# Main
def main():
    rospy.init_node('node_iot_ros_bridge_action_client')
    action_client = IotRosBridgeActionClient()

    goal_handle1 = action_client.send_goal("mqtt", "pub", action_client._config_mqtt_pub_
    action_client._goal_handles['1'] = goal_handle1
    rospy.loginfo("Goal #1 Sent")

    goal_handle2 = action_client.send_goal("mqtt", "sub", "eyrc/vb/mqtt/myTopic", "NA")
    action_client._goal_handles['2'] = goal_handle2
    rospy.loginfo("Goal #2 Sent")

    # rospy.sleep(1.0)
    # goal_handle1.cancel()

    rospy.spin()


if __name__ == '__main__':
    main()
```

Download

Run Command

Now this server do the following,

```
roscd pkg_iot_ros_bridge

cd srcipts

sudo chmod +x node_iot_ros_bridge_action_client.py

rosrun pkg_iot_ros_bridge node_iot_ros_bridge_action_client.py
```