Example #1: Action Server - ROS-MQTT Bridge

Aim

- To write a ROS Node which will act as Action Server. This Action Server should be able to process goals coming from one or more Action Clients.
- Create this Action Server Node in `pkg_iot_ros_bridge` ROS Package.

```
catkin_create_pkg pkg_iot_ros_bridge roscpp rospy std_msgs actionlib_msgs
```

- The name of the action use by this Action Server should be `/action_iot_ros`.
- The Action Server should act as a bridge between ROS and MQTT.
- This Server should take goals to either publish or subscribe to a MQTT Topic and should return the result as `True` if the goal is achieved or else `False`.
- 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
```

NOTE: Follow the steps provided in `2.5.8.3.1. Create a action message file` to create this action file.

- If an Action Client wants to publish on a MQTT Topic then the Goal message would carry the following.
  - protocol = `mqtt`
  - mode = `pub`
  - topic = `/eyrc/<unique_id>/ros_to_iot`
  - message = `Hello from ROS!`
- If an Action Client wants to subscribe to a MQTT Topic then the Goal message would carry the following.
  - protocol = `mqtt`
  - mode = `sub`
  - topic = `/eyrc/<unique_id>/iot_to_ros`
  - message = `NA`

- The Action Server should push the data coming from MQTT subscription to a ROS Topic called `/topic_iot_ros_bridge/mqtt/sub`.
- Communication on this ROS Topic should happen using a custom ROS Message `msgMqttSub.msg` with following content.

```
time timestamp
string topic
string message
```

Here `topic` will carry the MQTT Subscription Topic Name and `message` will carry the incoming message from MQTT Subscription.

NOTE: Steps to create a Custom ROS Message for a ROS Topic can be found this this section - `2.5.8.1.2. Example #1: Pub-Sub with Custom Message`.

- Upon receiving the Goal from the Client the Server should start a new thread to process the new incoming Goal.
- MQTT Server URL, Port, QoS and the ROS Topic on which MQTT Subscription message should be pushed, these things the Action Server should fetch from `config_ros_iot.yaml` file which should be loaded onto the Parameter Server.
- `config_iot_ros.yaml` should be present inside `config` folder inside `pkg_iot_ros_bridge`.

```
# config_iot_ros.yaml
# IoT Configuration
config_iot:
  mqtt:
    server_url: "broker.mqttdashboard.com"        # http://www.hivemq.com/demos/webs
    # server_url: "test.mosquitto.org"                    # Alternative to HiveMQ
    server_port: 1883
    topic_sub: "eyrc/xYzqLm/iot_to_ros"          # <unique_id> = xYzqLm
    topic_pub: "eyrc/xYzqLm/ros_to_iot"          # <unique_id> = xYzqLm
    qos: 0

    sub_cb_ros_topic: "/ros_iot_bridge/mqtt/sub"   # ROS nodes can listen to this to
```

- Create a separate python module called `iot` which can perform MQTT Tasks. The Action Server should import this module to perform MQTT tasks.

  NOTE: You can download this custom module along with the code using the download button given below.

Code

`node_iot_ros_bridge_action_server.py`

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

# ROS Node - Action Server - IoT ROS Bridge

import rospy
import actionlib
import threading

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
from pkg_iot_ros_bridge.msg import msgIotRosFeedback    # Message Class that is used for

from pkg_iot_ros_bridge.msg import msgMqttSub           # Message Class for MQTT Subscrip
```

```
from pyiot import iot                                   # Custom Python Module to perfrom


class IotRosBridgeActionServer:

    # Constructor
    def __init__(self):
        # Initialize the Action Server
        self._as = actionlib.ActionServer('/action_iot_ros',
                                          msgIotRosAction,
                                          self.on_goal,
                                          self.on_cancel,
                                          auto_start=False)

        '''
            * self.on_goal - It is the fuction pointer which points to a function which w
                             when the Action Server receives a Goal.

            * self.on_cancel - It is the fuction pointer which points to a function which
                             when the Action Server receives a Cancel Request.
        '''

        # Read and Store IoT Configuration data from Parameter Server
        param_config_iot = rospy.get_param('config_iot')
        self._config_mqtt_server_url = param_config_iot['mqtt']['server_url']
        self._config_mqtt_server_port = param_config_iot['mqtt']['server_port']
        self._config_mqtt_sub_topic = param_config_iot['mqtt']['topic_sub']
        self._config_mqtt_pub_topic = param_config_iot['mqtt']['topic_pub']
        self._config_mqtt_qos = param_config_iot['mqtt']['qos']
        self._config_mqtt_sub_cb_ros_topic = param_config_iot['mqtt']['sub_cb_ros_topic']
        print(param_config_iot)


        # Initialize ROS Topic Publication
        # Incoming message from MQTT Subscription will be published on a ROS Topic (/ros_
        # ROS Nodes can subscribe to this ROS Topic (/ros_iot_bridge/mqtt/sub) to get mes
        self._handle_ros_pub = rospy.Publisher(self._config_mqtt_sub_cb_ros_topic, msgMqt


        # Subscribe to MQTT Topic (eyrc/xYzqLm/iot_to_ros) which is defined in 'config_ic
        # self.mqtt_sub_callback() function will be called when there is a message from M
        ret = iot.mqtt_subscribe_thread_start(  self.mqtt_sub_callback,
                                                self._config_mqtt_server_url,
                                                self._config_mqtt_server_port,
                                                self._config_mqtt_sub_topic,
                                                self._config_mqtt_qos   )
        if(ret == 0):
            rospy.loginfo("MQTT Subscribe Thread Started")
        else:
            rospy.logerr("Failed to start MQTT Subscribe Thread")


        # Start the Action Server
        self._as.start()

        rospy.loginfo("Started ROS-IoT Bridge Action Server.")


    # This is a callback function for MQTT Subscriptions
    def mqtt_sub_callback(self, client, userdata, message):
        payload = str(message.payload.decode("utf-8"))

        print("[MQTT SUB CB] Message: ", payload)
        print("[MQTT SUB CB] Topic: ", message.topic)

        msg_mqtt_sub = msgMqttSub()
        msg_mqtt_sub.timestamp = rospy.Time.now()
        msg_mqtt_sub.topic = message.topic
        msg_mqtt_sub.message = payload

        self._handle_ros_pub.publish(msg_mqtt_sub)


    # This function will be called when Action Server receives a Goal
    def on_goal(self, goal_handle):
        goal = goal_handle.get_goal()

        rospy.loginfo("Received new goal from Client")
        rospy.loginfo(goal)

        # Validate incoming goal parameters
```

```
        if(goal.protocol == "mqtt"):

            if((goal.mode == "pub") or (goal.mode == "sub")):
                goal_handle.set_accepted()

                # Start a new thread to process new goal from the client (For Asynchronou
                # 'self.process_goal' - is the function pointer which points to a functic
                thread = threading.Thread(  name="worker",
                                            target=self.process_goal,
                                            args=(goal_handle,) )
                thread.start()

            else:
                goal_handle.set_rejected()
                return

        else:
            goal_handle.set_rejected()
            return


    # This function is called is a separate thread to process Goal.
    def process_goal(self, goal_handle):

        flag_success = False
        result = msgIotRosResult()

        goal_id = goal_handle.get_goal_id()
        rospy.loginfo("Processing goal : " + str(goal_id.id))

        goal = goal_handle.get_goal()


        # Goal Processing
        if(goal.protocol == "mqtt"):
            rospy.logwarn("MQTT")

            if(goal.mode == "pub"):
                rospy.logwarn("MQTT PUB Goal ID: " + str(goal_id.id))

                rospy.logwarn(goal.topic + " > " + goal.message)

                ret = iot.mqtt_publish( self._config_mqtt_server_url,
                                        self._config_mqtt_server_port,
                                        goal.topic,
                                        goal.message,
                                        self._config_mqtt_qos   )

                if(ret == 0):
                    rospy.loginfo("MQTT Publish Successful.")
                    result.flag_success = True
                else:
                    rospy.logerr("MQTT Failed to Publish")
                    result.flag_success = False

            elif(goal.mode == "sub"):
                rospy.logwarn("MQTT SUB Goal ID: " + str(goal_id.id))
                rospy.logwarn(goal.topic)

                ret = iot.mqtt_subscribe_thread_start(  self.mqtt_sub_callback,
                                                        self._config_mqtt_server_url,
                                                        self._config_mqtt_server_port,
                                                        goal.topic,
                                                        self._config_mqtt_qos   )
                if(ret == 0):
                    rospy.loginfo("MQTT Subscribe Thread Started")
                    result.flag_success = True
                else:
                    rospy.logerr("Failed to start MQTT Subscribe Thread")
                    result.flag_success = False

        rospy.loginfo("Send goal result to client")
        if (result.flag_success == True):
            rospy.loginfo("Succeeded")
            goal_handle.set_succeeded(result)
        else:
            rospy.loginfo("Goal Failed. Aborting.")
            goal_handle.set_aborted(result)

        rospy.loginfo("Goal ID: " + str(goal_id.id) + " Goal Processing Done.")
```

```
    # This function will be called when Goal Cancel request is send to the Action Server
    def on_cancel(self, goal_handle):
        rospy.loginfo("Received cancel request.")
        goal_id = goal_handle.get_goal_id()


# Main
def main():
    rospy.init_node('node_iot_ros_bridge_action_server')

    action_server = IotRosBridgeActionServer()

    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_server.py

rosrun pkg_iot_ros_bridge node_iot_ros_bridge_action_server.py
```