# Program Code -3

**UDP node Initialization using the defined topics**

**Global Imports and Configuration**

|  |
| --- |
| #!/usr/bin/env python3  import rclpy from rclpy.node import Node from std\_msgs.msg import Float32, String from mqtt\_offered\_client.mqtt\_offered\_client import PriorityMQTTClient import socket import struct import yaml import os |

**Class: UDPServerNode**

|  |
| --- |
| class UDPServerNode(Node): |

**Function: to Initialize the mqtt and UDP configuration**

|  |
| --- |
| def \_\_init\_\_(self):  super().\_\_init\_\_('udp\_server')   # Load configuration from YAML file  config\_file = os.getenv('CONFIG\_FILE', 'configure\_mqtt.yaml')  module\_name = os.getenv('MODULE\_NAME', 'grid') # Default to 'grid' if not set  with open(config\_file, 'r') as file:  config = yaml.safe\_load(file)   # UDP Configuration  self.udp\_ip = config['udp']['modules'][module\_name]['source\_ip']  self.udp\_port = config['udp']['port']  self.destination\_ips = config['udp']['modules'][module\_name]['destination\_ips']  self.destination\_port = config['udp']['port']   # Initialize UDP socket  self.sock = socket.socket(socket.AF\_INET, socket.SOCK\_DGRAM)  self.sock.bind((self.udp\_ip, self.udp\_port))  self.get\_logger().info(f"UDP Server listening on {self.udp\_ip}:{self.udp\_port}")   # Send initial message to all destination IPs  message = f"PoH says oh oh oh from {module\_name}".encode()  for dest\_ip in self.destination\_ips:  if dest\_ip != self.udp\_ip: # Avoid sending to self  self.sock.sendto(message, (dest\_ip, self.destination\_port))  self.get\_logger().info(f"Sent message to {dest\_ip}:{self.destination\_port}")   # Initialize PriorityMQTTClient  self.mqtt\_client = PriorityMQTTClient(  broker=config['mqtt']['broker'],  port=config['mqtt']['port']  )   **# Topic mappings from MQTT configuration**  self.topic\_mappings = {  '/ros2mqtt/grid/frequency/reference': {'mqtt\_topic': 'phil/grid/frequency/reference', 'qos': 2, 'type': Float32},  '/ros2mqtt/grid/frequency/measured': {'mqtt\_topic': 'phil/grid/frequency/measured', 'qos': 2, 'type': Float32},  '/ros2mqtt/grid/voltage/phase1': {'mqtt\_topic': 'phil/grid/voltage/phase1', 'qos': 1, 'type': Float32},  '/ros2mqtt/supercap/mode': {'mqtt\_topic': 'phil/supercap/mode', 'qos': 0, 'type': String},  '/ros2mqtt/supercap/voltage/phase1': {'mqtt\_topic': 'phil/supercap/voltage/phase1', 'qos': 1, 'type': Float32},  '/ros2mqtt/battery/mode': {'mqtt\_topic': 'phil/battery/mode', 'qos': 0, 'type': String},  '/ros2mqtt/battery/voltage/phase1': {'mqtt\_topic': 'phil/battery/voltage/phase1', 'qos': 1, 'type': Float32},  '/ros2mqtt/battery/power/active': {'mqtt\_topic': 'phil/battery/power/active', 'qos': 1, 'type': Float32},  '/ros2mqtt/fuelcell/mode': {'mqtt\_topic': 'phil/fuelcell/mode', 'qos': 0, 'type': String},  '/ros2mqtt/fuelcell/voltage/phase1': {'mqtt\_topic': 'phil/fuelcell/voltage/phase1', 'qos': 1, 'type': Float32},  '/ros2mqtt/fuelcell/power/active': {'mqtt\_topic': 'phil/fuelcell/power/active', 'qos': 1, 'type': Float32},  '/ros2mqtt/component/status': {'mqtt\_topic': 'phil/component/status', 'qos': 2, 'type': String},  '/ros2mqtt/component/logs': {'mqtt\_topic': 'phil/component/logs', 'qos': 0, 'type': String},  '/ros2mqtt/alert/grid\_failure': {'mqtt\_topic': 'phil/alert/grid\_failure', 'qos': 2, 'type': String},  '/ros2mqtt/connectivity/status': {'mqtt\_topic': 'phil/connectivity/status', 'qos': 0, 'type': String}  }   **# Create publishers and subscriptions**  self.publishers = {}  self.subscriptions = {}  for ros\_topic, config in self.topic\_mappings.items():  self.publishers[ros\_topic] = self.create\_publisher(config['type'], ros\_topic, 10)  callback = self.create\_callback(config['mqtt\_topic'], config['qos'])  self.subscriptions[ros\_topic] = self.create\_subscription(  config['type'], ros\_topic, callback, 10  )   # Timer for receiving UDP messages  self.timer = self.create\_timer(0.01, self.receive\_message) |

**Function: create\_callback**

|  |
| --- |
| def create\_callback(self, mqtt\_topic, qos):  """Create a callback function for a specific MQTT topic and QoS.""" |

**Function: callback**

|  |
| --- |
| def callback(msg):  self.mqtt\_client.publish(mqtt\_topic, str(msg.data), qos=qos)  self.get\_logger().info(f"Published to MQTT topic {mqtt\_topic}: {msg.data}")  return callback |

**Function: receive\_message**

|  |
| --- |
| def receive\_message(self):  try:  # Receive UDP data (4 bytes for float)  data, addr = self.sock.recvfrom(4)  self.get\_logger().info(f"Received raw data from {addr}: {data}")   # Interpret data as a float  try:  received\_float = struct.unpack('f', data)[0]  # Publish to ROS 2 topic (assuming supercap/voltage/phase1 for now)  ros\_topic = '/ros2mqtt/supercap/voltage/phase1'  float\_msg = Float32()  float\_msg.data = received\_float  if ros\_topic in self.publishers:  self.publishers[ros\_topic].publish(float\_msg)  self.get\_logger().info(f"Published to ROS 2 topic {ros\_topic}: {received\_float}")  # Forward data to other modules  for dest\_ip in self.destination\_ips:  if dest\_ip != self.udp\_ip and dest\_ip != addr[0]: # Avoid sending to self or sender  self.sock.sendto(data, (dest\_ip, self.destination\_port))  self.get\_logger().info(f"Forwarded data to {dest\_ip}:{self.destination\_port}")  else:  self.get\_logger().warn(f"No publisher for ROS 2 topic {ros\_topic}")  except struct.error:  self.get\_logger().warn(f"Invalid float data from {addr}: {data}")   except socket.timeout:  pass # No data received, continue  except Exception as e:  self.get\_logger().error(f"Error receiving data: {e}") |

**Function: destroy\_node**

|  |
| --- |
| def destroy\_node(self):  self.get\_logger().info("Shutting down UDP server...")  self.mqtt\_client.stop()  self.sock.close()  super().destroy\_node() |

**Function: main**

|  |
| --- |
| def main(args=None):  rclpy.init(args=args)  node = UDPServerNode()  try:  rclpy.spin(node)  except KeyboardInterrupt:  node.get\_logger().info("Keyboard Interrupt detected, shutting down.")  finally:  node.destroy\_node()  rclpy.shutdown()  if \_\_name\_\_ == '\_\_main\_\_':  main() |