**Name: Nara Cheyklin ID: 6538097021**

1. Screenshot of ros2 interface show ros2_text_encoder/srv/TextEncode

```
root@Thongdang:~/ros2_ws# ros2 interface show ros2_text_encoder/srv/TextEncode
string text
string scheme
---
builtin_interfaces/Time stamp
        int32 sec
        uint32 nanosec
string encoded
```

2. Screenshot of ros2 node info <node_name>

```
root@Thongdang:~/ros2_ws# ros2 node info /text_encode_server
/text_encode_server
  Subscribers:

  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /text_encode: ros2_text_encoder/srv/TextEncode
    /text_encode_server/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /text_encode_server/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /text_encode_server/get_parameters: rcl_interfaces/srv/GetParameters
    /text_encode_server/list_parameters: rcl_interfaces/srv/ListParameters
    /text_encode_server/set_parameters: rcl_interfaces/srv/SetParameters
    /text_encode_server/set_parameters_atomically: rcl_interfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

3. Screenshot of service call result (showing encoded output)

```
root@Thongdang:~/ros2_ws# ros2 service call /text_encode ros2_text_encoder/srv/TextEncode "text: 'hello'
scheme: 'base64'"
requester: making request: ros2_text_encoder.srv.TextEncode_Request(text='hello', scheme='base64')

response:
ros2_text_encoder.srv.TextEncode_Response(stamp=builtin_interfaces.msg.Time(sec=1761045948, nanosec=570432312), encoded='aGVsbG8=')
```

4. Sourcecode:

4.1 ) text_encode_server.py

```python
from ros2_text_encoder.srv import TextEncode
import rclpy
from rclpy.node import Node
import base64, codecs, hashlib, urllib

class TextEncodeServer(Node):

    def __init__(self):
        super().__init__('text_encode_server')
        self.srv = self.create_service(TextEncode, 'text_encode', self.text_encode_callback )

    def text_encode_callback(self, request, response):
        text = request.text
        scheme = request.scheme
        response.encoded = self.encode_text(text, scheme)
        response.stamp = self.get_clock().now().to_msg()
        return response

    def encode_text(self, text: str, scheme: str) -> str:

        s = (scheme or '').strip().lower()
        if s == 'base64':
            return base64.b64encode(text.encode('utf-8')).decode('utf-8')
        elif s == 'rot13':
            return codecs.encode(text, 'rot_13')
        elif s == 'sha256':
            return hashlib.sha256(text.encode('utf-8')).hexdigest()
        elif s == 'url':
            return urllib.parse.quote(text, safe='')
        if s.startswith('xor:'):
            try:
                key = int(s.split(':', 1)[1])
            except Exception:
                raise ValueError("xor scheme must be 'xor:<int>' เช่น xor:42")
            bs = bytes([b ^ (key & 0xFF) for b in text.encode('utf-8')])
            return base64.b64encode(bs).decode('utf-8')
        raise ValueError(f'Unknown scheme: {scheme} (รองรับ: base64, rot13, sha256, url, xor:<int>)')
```

```python
def main():
    rclpy.init()
    text_encode_server = TextEncodeServer()
    rclpy.spin(text_encode_server)
    text_encode_server.destroy_node()
    rclpy.shutdown()

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

## 4.2) turtle_sub.py

```python
import rclpy
import math
from rclpy.node import Node
from turtlesim.msg import Pose
from geometry_msgs.msg import Vector3, Quaternion
# from ros2_text_encoder.msg import Turtle
from ros2_text_encoder.srv import Turtlesrv


def quat_from_yaw(yaw: float) -> Quaternion:
    """Make a quaternion (x=0,y=0) from a yaw (theta) angle."""
    q = Quaternion()
    q.x = 0.0
    q.y = 0.0
    q.z = math.sin(yaw / 2.0)
    q.w = math.cos(yaw / 2.0)
    return q
class TurtleSub(Node):

    def __init__(self):
        super().__init__('turtle_sub')
        self.latest=None
        self.subscription = self.create_subscription(
            Pose,
            '/turtle1/pose',
            self.pose_callback,
            10
        )
        self.srv = self.create_service(Turtlesrv, 'turtlesrv', self.handle_callback)
```

```python
    def pose_callback(self, msg: Pose):
        self.latest= msg
    def handle_callback(self, request, response):
        if self.latest is None:
            # Haven't received any pose yet; return zeros but don't block
            self.get_logger().warn("No /turtle1/pose received yet; returning zeros.")
            response.position = Vector3(x=0.0, y=0.0, z=0.0)
            response.orientation = Quaternion(x=0.0, y=0.0, z=0.0, w=1.0)
            return response
        response.position = Vector3(x=self.latest.x, y=self.latest.y, z=0.0)
        response.orientation = quat_from_yaw(self.latest.theta)
        return response

def main():
    rclpy.init()
    turtle_sub = TurtleSub()
    rclpy.spin(turtle_sub)
    turtle_sub.destroy_node()
    rclpy.shutdown

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

Command:

1.

```
root@Thongdang:~# ros2 run turtlesim turtlesim_node
QStandardPaths: wrong permissions on runtime directory /run/user/0/, 0755 instead of 0700
[INFO] [1761077544.820045859] [turtlesim]: Starting turtlesim with node name /turtlesim
[INFO] [1761077544.823942821] [turtlesim]: Spawning turtle [turtle1] at x=[5.544445], y=[5.544445], theta=[0.000000]
```

2. `root@Thongdang:~# ros2 run ros2_text_encoder turtle_sub`

3. Service call twice: first time at the spawn point, secound time after a movement with turtle_teleop_key

```
ros2 service call /turtlesrv ros2_text_encoder/srv/Turtlesrv "{}"
requester: making request: ros2_text_encoder.srv.Turtlesrv_Request()

response:
ros2_text_encoder.srv.Turtlesrv_Response(position=geometry_msgs.msg.Vector3(x=5.544444561004639, y=5.544444561004639, z=0.0), orientation=ge
ometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.0, w=1.0))
```

```
root@Thongdang:~# ros2 service call /turtlesrv ros2_text_encoder/srv/Turtlesrv "{}"
requester: making request: ros2_text_encoder.srv.Turtlesrv_Request()

response:
ros2_text_encoder.srv.Turtlesrv_Response(position=geometry_msgs.msg.Vector3(x=3.5566816329956055, y=6.406960487365723, z=0.0), orientation=g
eometry_msgs.msg.Quaternion(x=0.0, y=0.0, z=0.5175627294434486, w=0.8556452659198481))
```

Turtle Final Position