

# 05.01 스터디

## ROS2 심화기능2

- RQt Plugin
- ROS2 Lifecycle
- ROS2 security

### RQt

- ROS 2 시스템을 시각적으로 조작(디버깅/모니터링)하기 위한 GUI 툴킷
- Qt 기반으로 구성되며 플러그인 방식으로 기능을 확장 가능

RQt Plugin: rqt라는 기본 GUI 프레임워크에 패키지 형태의 기능을 확장

### RQt 플러그인 패키지

패키지 이름	설명
<b>rqt_gui</b>	여러 rqt 위젯을 단일 창에 도킹할 수 있는 위젯 패키지 rqt 명령어를 실행하면 이게 작동함
<b>rqt_gui_cpp</b>	C++ 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API 제공
<b>rqt_gui_py</b>	Python 클라이언트 라이브러리를 사용하여 제작할 수 있는 RQt GUI 플러그인 API 제공
<b>rqt_py_common</b>	Python으로 작성된 RQt 플러그인에서 공용으로 사용되는 기능을 모 듈로 제공하는 패키지
<b>rqt_common_plugins</b>	rqt_action, rqt_bag 등 20여개의 RQt 플러그인을 포함하는 메타패 키지
<b>qt_gui_core</b>	qt_gui, qt_gui_cpp, qt_gui_py_common, qt_gui_app, qt_dotgraph 등을 담은 메타패키지
<b>python_qt_binding</b>	QtCore, QtGui, QtWidgets 등을 사용할 때 Python 언어 기반의 Qt API를 제공하는 바인딩 패키지

## ▼ (참고) Qt, PyQt, RQt

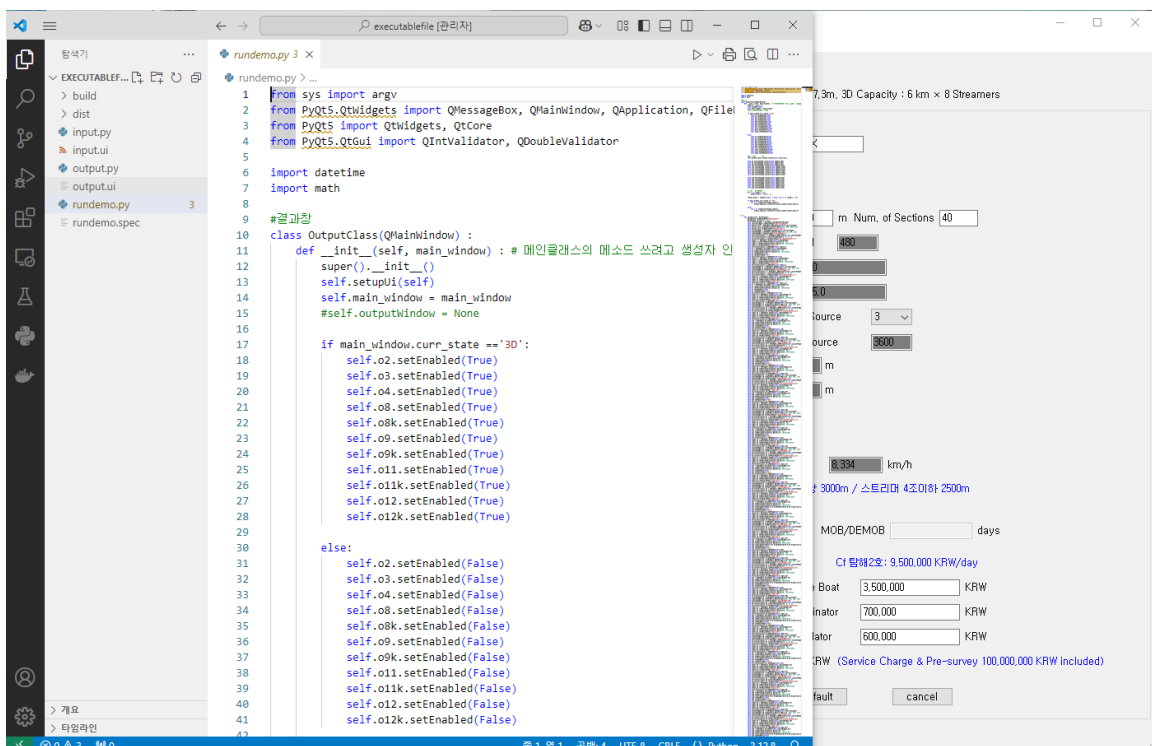
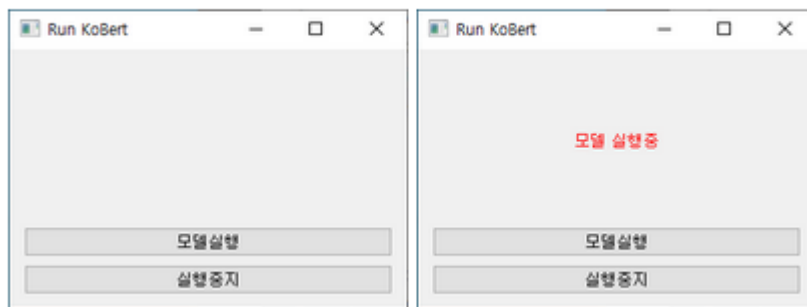
### Qt

C++로 개발된 크로스 플랫폼 애플리케이션 개발 프레임워크. 다양한 운영체제에서 동일한 코드로 작동하는 프로그램을 만들 수 있음. 풍부한 GUI 위젯과 도구들을 제공하여 사용자 친화적인 인터페이스 개발에 많이 사용됩니다.

### PyQt

파이썬에서 Qt 프레임워크를 사용할 수 있게 해주는 라이브러리

#### • Model

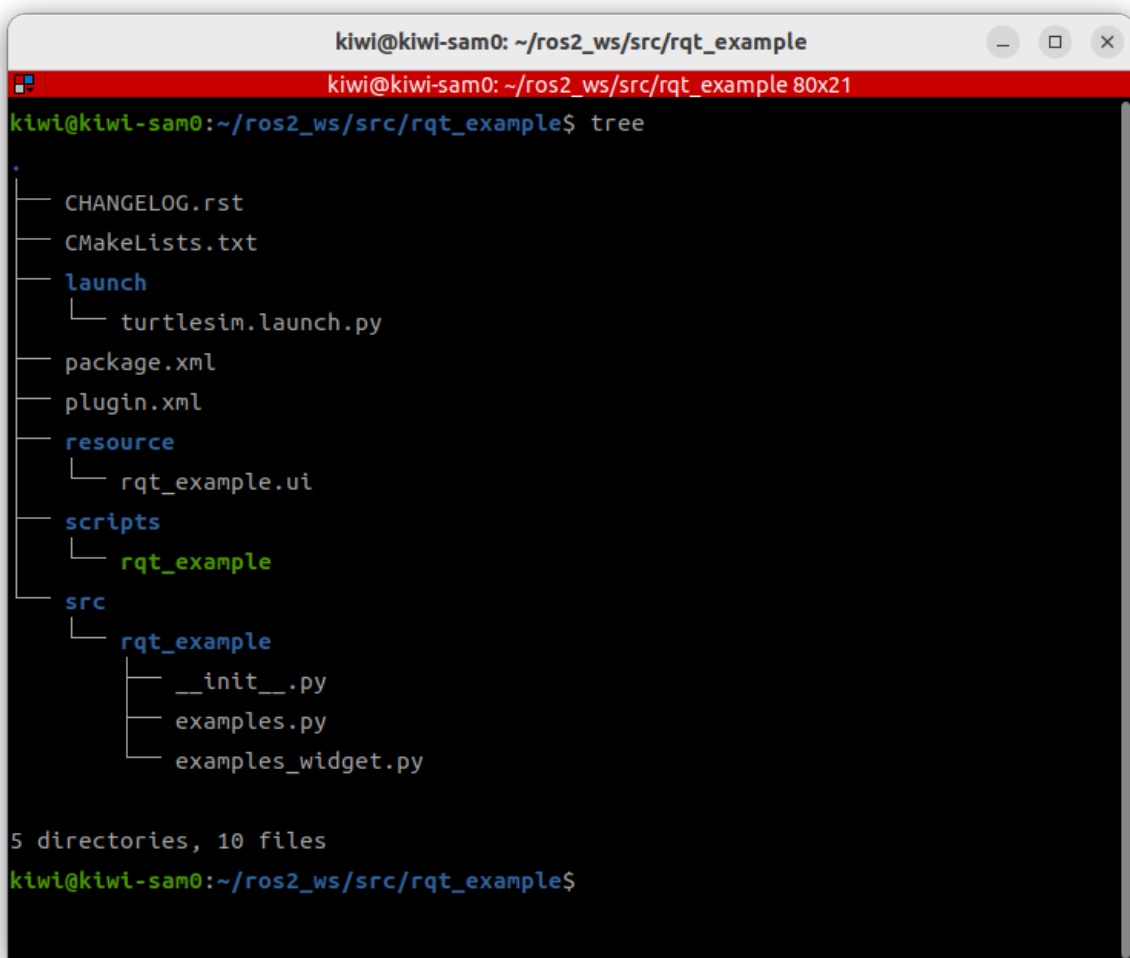


## RQt

ROS 생태계 내에서 특정한 목적을 가지고 개발된 프레임워크

단순 바인딩처럼 Qt의 모든 기능을 그대로 파이썬에서 사용할 수 있게 해주는 것이 아니라, ROS 환경에 맞춰서 GUI 도구를 쉽게 만들 수 있도록 여러 가지 편의 기능과 구조를 제공

## rqt\_example 구조



```
kiwi@kiwi-sam0: ~/ros2_ws/src/rqt_example
kiwi@kiwi-sam0: ~/ros2_ws/src/rqt_example 80x21
kiwi@kiwi-sam0:~/ros2_ws/src/rqt_example$ tree
.
├── CHANGELOG.rst
├── CMakeLists.txt
├── launch
│   └── turtlesim.launch.py
├── package.xml
├── plugin.xml
├── resource
│   └── rqt_example.ui
├── scripts
│   └── rqt_example
├── src
│   └── rqt_example
│       ├── __init__.py
│       ├── examples.py
│       └── examples_widget.py
└──

5 directories, 10 files
kiwi@kiwi-sam0:~/ros2_ws/src/rqt_example$
```

## CHANGELOG.rst

- 패키지의 버전별 변경 이력

## CMakeList.txt

- 패키지 빌드 설정

## src/rqt\_example/\_\_init\_\_.py

- 비어있는 스크립트
- 파이썬 패키지로 인식시키기 위한 용도

## src/rqt\_example/examples.py

- `Plugin` 클래스 상속하여 UI 초기화 및 위젯 등록
- 종료 시 `shutdown_plugin()` 실행

```
from rqt_example.examples_widget import ExamplesWidget
from rqt_gui_py.plugin import Plugin
```

```
class Examples(Plugin):
```

```
    def __init__(self, context):
        super(Examples, self).__init__(context)
        self.setObjectName('RQt example')
        self.widget = ExamplesWidget(context.node)
        serial_number = context.serial_number()
        if serial_number > 1:
            self.widget.setWindowTitle(self.widget.windowTitle() + ' ({0})'.format(s
            context.add_widget(self.widget)
```

```
    def shutdown_plugin(self):
        print('Shutdown the RQt example.')
        self.widget.shutdown_widget()
```

## src/rqt\_example/examples\_widget.py

- ui 파일을 불러와 GUI 구성을 적용
- ROS 2의 토픽 퍼블리셔/서브스크라이버 및 서비스 서버/클라이언트 생성
- 버튼/키보드 입력으로 속도 제어 ( `cmd_vel` 발행)
- 수신한 속도값을 슬라이더와 LCD에 실시간 표시
- 라디오 버튼으로 LED 상태 제어 서비스 호출

### ▼ examples\_widget.py

```
#!/usr/bin/env python3

import os

from ament_index_python.resources import get_resource
from geometry_msgs.msg import Twist
from python_qt_binding import loadUi
from python_qt_binding.QtCore import Qt
from python_qt_binding.QtCore import QTimer
from python_qt_binding.QtGui import QKeySequence
from python_qt_binding.QtWidgets import QShortcut
from python_qt_binding.QtWidgets import QWidget

import rclpy
from rclpy.qos import QoSProfile
from std_srvs.srv import SetBool

class ExamplesWidget(QWidget):

    def __init__(self, node):
        super(ExamplesWidget, self).__init__()
        self.setObjectName('ExamplesWidget')

        self.node = node
```

```

# 속도 출력 주기 및 UI 갱신 주기(ms)
self.REDRAW_INTERVAL = 30
self.PUBLISH_INTERVAL = 100

# 슬라이더와 다이얼 표시값을 위한 스케일 조절
self.CMD_VEL_X_FACTOR = 1000.0
self.CMD_VEL_YAW_FACTOR = -10.0

# UI 파일 로드
pkg_name = 'rqt_example'
ui_filename = 'rqt_example.ui'
topic_name = 'cmd_vel'
service_name = 'led_control'

_, package_path = get_resource('packages', pkg_name)
ui_file = os.path.join(package_path, 'share', pkg_name, 'resource', ui_
loadUi(ui_file, self)

# 퍼블리시/서브스크라이브할 Twist 메시지 초기화
self.pub_velocity = Twist()
self.pub_velocity.linear.x = 0.0
self.pub_velocity.angular.z = 0.0
self.sub_velocity = Twist()
self.sub_velocity.linear.x = 0.0
self.sub_velocity.angular.z = 0.0

# 초기 슬라이더/LCD 값 설정
self.slider_x.setValue(0)
self.lcd_number_x.display(0.0)
self.lcd_number_yaw.display(0.0)

# 퍼블리셔, 서브스크라이버, 서비스 생성
qos = QoSProfile(depth=10)
self.publisher = self.node.create_publisher(Twist, topic_name, qos)
self.subscriber = self.node.create_subscription(Twist, topic_name, se
self.service_server = self.node.create_service(SetBool, service_name
self.service_client = self.node.create_client(SetBool, service_name)

```

```

# 타이머 설정 - 주기적으로 퍼블리시 및 UI 업데이트
self.publish_timer = QTimer(self)
self.publish_timer.timeout.connect(self.send_velocity)
self.publish_timer.start(self.PUBLISH_INTERVAL)

self.update_timer = QTimer(self)
self.update_timer.timeout.connect(self.update_indicators)
self.update_timer.start(self.REDRAW_INTERVAL)

# 키보드/버튼 입력과 콜백 연결 (속도 조절)
self.push_button_w.pressed.connect(self.increase_linear_x)
self.push_button_x.pressed.connect(self.decrease_linear_x)
self.push_button_a.pressed.connect(self.increase_angular_z)
self.push_button_d.pressed.connect(self.decrease_angular_z)
self.push_button_s.pressed.connect(self.set_stop)

# 키보드 단축키 할당
self.push_button_w.setShortcut('w')
self.push_button_x.setShortcut('x')
self.push_button_a.setShortcut('a')
self.push_button_d.setShortcut('d')
self.push_button_s.setShortcut('s')

# space 키로 정지 동작도 할 수 있도록 설정
self.shortcut_space = QShortcut(QKeySequence(Qt.Key_Space), self)
self.shortcut_space.setContext(Qt.ApplicationShortcut)
self.shortcut_space.activated.connect(self.push_button_s.pressed)

# LED 제어 라디오버튼 클릭 시 서비스 요청
self.radio_button_led_on.clicked.connect(self.call_led_service)
self.radio_button_led_off.clicked.connect(self.call_led_service)

# LED 제어용 단축키 등록
self.radio_button_led_on.setShortcut('o')
self.radio_button_led_off.setShortcut('f')

def get_velocity(self, msg):
    # 서브스크라이버 콜백 - 수신된 속도 저장

```

```

self.sub_velocity = msg

def set_led_status(self, request, response):
    # 서비스 서버 - 요청에 따라 LED 상태 반영
    if request.data:
        self.push_button_led_status.setText('ON')
        self.push_button_led_status.setStyleSheet('color: rgb(255, 170, 0);')
        response.success = True
        response.message = 'LED ON'
    elif not request.data:
        self.push_button_led_status.setText('OFF')
        self.push_button_led_status.setStyleSheet('')
        response.success = True
        response.message = 'LED OFF'
    else:
        response.success = False
    return response

# 퍼블리시할 선속도/회전속도 조절 함수들
def increase_linear_x(self):
    self.pub_velocity.linear.x += 0.1

def decrease_linear_x(self):
    self.pub_velocity.linear.x -= 0.1

def increase_angular_z(self):
    self.pub_velocity.angular.z += 0.1

def decrease_angular_z(self):
    self.pub_velocity.angular.z -= 0.1

def set_stop(self):
    self.pub_velocity.linear.x = 0.0
    self.pub_velocity.angular.z = 0.0

def call_led_service(self):
    # 서비스 클라이언트 호출 - 라디오버튼 상태에 따라 요청 전송
    request = SetBool.Request()

```



```

if self.radio_button_led_on.isChecked():
    request.data = True
elif self.radio_button_led_off.isChecked():
    request.data = False

# 서비스가 뜰 때까지 최대 5번 대기
wait_count = 1
while not self.service_client.wait_for_service(timeout_sec=0.5):
    if wait_count > 5:
        return
    self.node.get_logger().error('Service not available #{0}'.format(wait_count))
    wait_count += 1

# 비동기 요청
future = self.service_client.call_async(request)

while rclpy.ok():
    if future.done():
        if future.result() is not None:
            response = future.result()
            self.node.get_logger().info(
                'Result of service call: {0}'.format(response.message))
        else:
            self.node.get_logger().error('Error calling service')
        break

def send_velocity(self):
    # Twist 메시지 생성 후 퍼블리시
    twist = Twist()
    twist.linear.x = self.pub_velocity.linear.x
    twist.linear.y = 0.0
    twist.linear.z = 0.0
    twist.angular.x = 0.0
    twist.angular.y = 0.0
    twist.angular.z = self.pub_velocity.angular.z
    self.publisher.publish(twist)

```

```
def update_indicators(self):
    # 수신된 속도값을 UI 요소로 표시
    self.slider_x.setValue(int(self.sub_velocity.linear.x * self.CMD_VEL_X_F
    self.dial_yaw.setValue(int(self.sub_velocity.angular.z * self.CMD_VEL_`
    self.lcd_number_x.display(self.sub_velocity.linear.x)
    self.lcd_number_yaw.display(self.sub_velocity.angular.z)

def shutdown_widget(self):
    # 위젯 종료 시 리소스 해제
    self.update_timer.stop()
    self.publish_timer.stop()
    self.node.destroy_client(self.service_client)
    self.node.destroy_service(self.service_server)
    self.node.destroy_subscription(self.subscriber)
    self.node.destroy_publisher(self.publisher)
```

## package.xml

- ROS 2 패키지의 메타데이터와 의존성, 플러그인 등록 정보를 정의
- RQt에 이 패키지에서 제공하려는 플러그인을 추가하는 기능

```
<export>
  <build_type>ament_cmake</build_type>
  <rqt_gui plugin="${prefix}/plugin.xml"/>
</export>
```

<export>태그: ROS 2에서 이 패키지가 RQt 플러그인이라는 것을 (시스템에) 명시

## plugin.xml

- Group 태그가 메뉴의 세부 항목이 되며 <label>, <icon>, <statustip>이 해당 RQt 플러그인의 속성이 됨

```
<group>
  <label>Visualization</label>
  <icon type="theme">folder</icon>
  <statustip>Plugins related to visualization</statustip>
```

```

</group>
<label>Viewer</label>
<icon type="theme">utilities-system-monitor</icon>
<statustip>A plugin visualizing messages and services values</statustip>

```

## resource/rqt\_example.ui

- RQt 플러그인의 GUI 레이아웃을 정의한 Qt Designer용 XML 파일
- 수작업(코딩)으로 작업하는 대신 qtcreator에서 손쉽게 구성할 수 있다.

### \*qtcreator

Qt 통합 개발 환경

GUI 디자인뿐만 아니라 코드 편집, 빌드, 디버깅 등 소프트웨어 개발의 전반적인 과정을 지원

Qt Designer의 기능이 내장되어 있음

```
qtcreator ~/ros2_ws/src/rqt_example/resource/rqt_example.ui
```

### \*.ui파일

#### ui파일 형식이 xml인데도 확장자가 .ui인 이유?

ui 파일은 Qt Designer에서 만든 GUI 디자인 파일로, 내부는 XML 형식.

위젯 구조, 속성, 레이아웃 정보를 담고 있으며, 확장자 ui는 Qt 도구들이 인식할 수 있게 해 줌.

loadUi 함수는 ui 파일을 읽어 파이썬 위젯에 GUI를 동적으로 적용함.

확장자가 ui가 아니면 loadUi에서 읽히지 않음.

```

## examples_widget.py
pkg_name = 'rqt_example'
ui_filename = 'rqt_example.ui'
topic_name = 'cmd_vel'
service_name = 'led_control'

_, package_path = get_resource('packages', pkg_name)

```

```
ui_file = os.path.join(package_path, 'share', pkg_name, 'resource', ui_filename)
loadUi(ui_file, self)
```

## launch/turtlesim.launch.py

- `/turtle1/rqt_example` , `/turtlesim` 노드 실행
- 터미널에 안내 메시지(로그) 출력

```
from launch import LaunchDescription
from launch.actions import LogInfo
from launch_ros.actions import Node

def generate_launch_description():
    return LaunchDescription([
        LogInfo(msg=['Execute the rqt_example with turtlesim node.']),

        Node(
            namespace='turtle1',
            package='rqt_example',
            executable='rqt_example',
            name='rqt_example',
            output='screen'),

        Node(
            package='turtlesim',
            executable='turtlesim_node',
            name='turtlesim',
            output='screen')
    ])
```

## scripts/rqt\_example

- RQt 플러그인을 단독 실행할 수 있도록 설정된 진입점

```
#!/usr/bin/env python3
import sys
from rqt_gui.main import Main
from rqt_example.examples import Examples

plugin = "rqt_example.examples.Examples"
main = Main(filename=plugin)
sys.exit(main.main(standalone=plugin))
```

### \*.py 확장자 없어도 되나?

ROS 2에서 `setup.py` 또는 `CMakeLists.txt` 에 `install(PROGRAMS ...)` 로 등록한 스크립트는 실행 파일로 취급되고, `ros2 run <패키지명> <실행명>` 명령에서 `.py` 확장자를 요구하지 않음

```
## CMakeLists.txt

install(PROGRAMS
  scripts/rqt_example
  DESTINATION lib/${PROJECT_NAME}
)
```

### 동작과정 정리

```
ros2 launch rqt_example turtlesim.launch.py
[ launch file ]
↓
[turtlesim_node] + [rqt → plugin.xml]
↓
[scripts/rqt_example] (entry)
↓
[examples.py → Examples (Plugin)]
↓
[examples_widget.py → ExamplesWidget (UI + ROS)]
↓
[rqt_example.ui (Qt 위젯 + 레이아웃)]
↓
```

[Publisher, Subscriber, Service 작동]

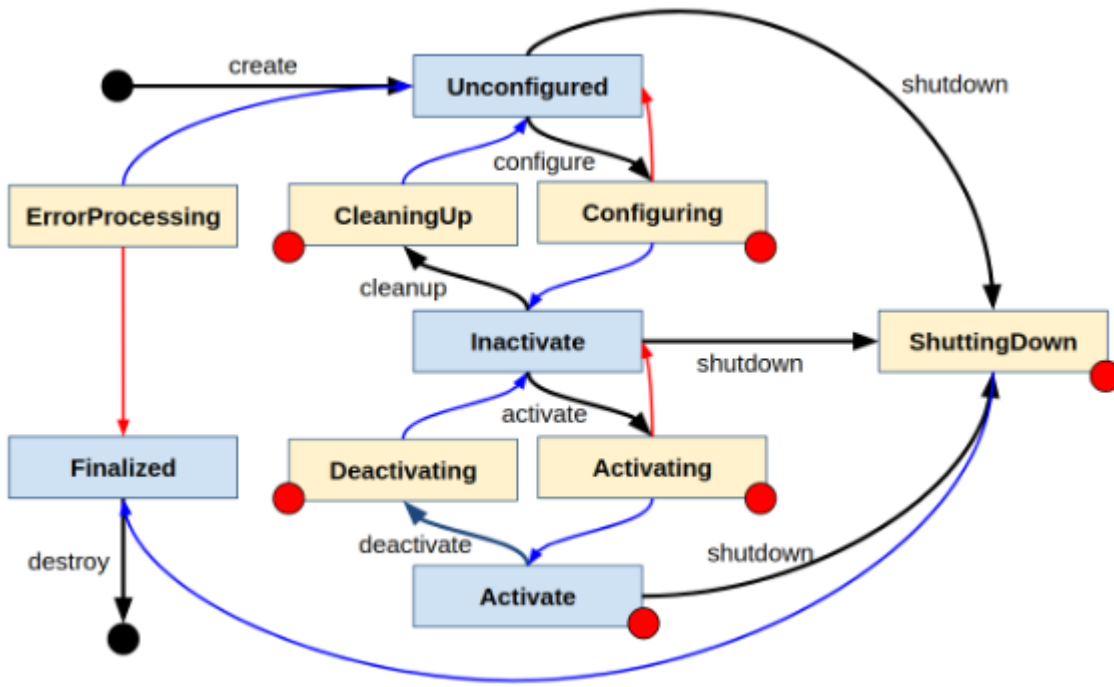


[turtlesim이 거북이 움직임 반영]

1. launch 파일을 실행하면 turtlesim과 rqt GUI가 함께 켜진다.
2. rqt는 plugin.xml을 읽어 어떤 플러그인을 로딩할지 확인한다.
3. plugin.xml에 지정된 실행 파일을 통해 Python 코드가 시작된다.
4. examples.py에서 플러그인 클래스를 실행하고, 실제 UI와 기능을 가진 examples\_widget.py를 띄운다.
5. examples\_widget.py는 rqt\_example.ui를 불러와 GUI를 구성하고, 퍼블리셔·서브스 크라이버·서비스 클라이언트를 설정한다.
6. 사용자가 버튼이나 키를 누르면 ROS 토픽과 서비스가 작동하며, turtlesim에서 거북이 움직임으로 반영된다.

## ROS2 Lifecycle

- ROS2에서는 노드의 상태 관리를 위해 Lifecycle 인터페이스 제공
- 노드를 동작 이전 상태에서 설정, 안전하게 중단 및 재설정 가능하게 함



## 주요 상태

- Unconfigured: 노드가 생성된 직후의 상태, 에러 발생 이후 다시 조정될 수 있는 상태
- Inactivate: 노드가 동작을 수행하지 않는 상태. 파라미터 등록, 토픽 발간과 구독 추가 삭제 등을 (재)구성 할 수 있는 상태
- Activate: 노드가 동작을 수행하는 상태.
- Finalized: 노드가 메모리에서 해제되기 직전 상태 노드가 파괴되기 전 디버깅이나 내부 검사를 진행할 수 있는 상태

## 전환 상태

- Configuring: 노드를 구성하기 위해 필요한 설정 수행
- CleaningUp: 노드가 처음 생성되었을 때 상태와 동일하게 만드는 과정 수행
- Activating: 노드가 동작을 수행하기 전 마지막 준비 과정 수행
- Deactivating : 노드가 동작을 수행하기 전으로 돌아가는 과정 수행
- ShuttingDown: 노드가 파괴되기 전 필요한 과정 수행
- ErrorProcessing: 사용자 코드가 동작되는 상태에서 발생하는 에러를 해결하기 위한 과정 수행

## 전환

- Create: 노드를 생성하고 초기 상태로 설정
- Configure: 노드를 구성하여 준비 상태로 만듦
- Cleanup: 노드를 초기화하여 이전 상태로 되돌림
- Activate: 노드를 활성화하여 기능을 수행할 수 있게 함
- Deactivate: 노드를 비활성화하여 동작을 멈춤
- Shutdown: 노드를 안전하게 종료하는 과정
- Destroy: 노드를 메모리에서 완전히 제거

## 상태 전이 흐름

Unconfigured → (configure) → Inactive → (activate) → Active  
Active → (deactivate) → Inactive → (cleanup) → Unconfigured

## 실습

서비스 호출을 통해 talker 노드의 상태 변경하기

```
# 라이프사이클을 따르는 퍼블리셔 노드
ros2 run lifecycle lifecycle_talker

# lifecycle_talker가 발행하는 토픽을 구독하는 노드
ros2 run lifecycle lifecycle_listener

# talker 상태 전환
ros2 run lifecycle lifecycle_service_client
```

## ROS2 security

Security는 SROS의 유틸리티로, DDS-Security를 ROS2에서 사용하기 위해 필요한 도구를 모아둔 것



```
# 보안키 저장소 생성
ros2 security create_keystore demo_keystore

# 보안키 생성
# ros2 security create_enclave demo_keystore /talker_listener/talker
ros2 security create_enclave demo_keystore /talker_listener/talker
ros2 security create_enclave demo_keystore /talker_listener/listener

# 환경변수 설정
export ROS_SECURITY_KEYSTORE=~/.ros2_demo/demo_keystore
export ROS_SECURITY_ENABLE=true
export ROS_SECURITY_STRATEGY=Enforce
```

## 환경변수 3가지

환경 변수	설명
<code>ROS_SECURITY_KEYSTORE</code>	보안설정 파일 저장소 경로
<code>ROS_SECURITY_ENABLE</code>	보안 활성화 여부 true, false (Default는 false)
<code>ROS_SECURITY_STRATEGY</code>	보안 설정 방법 Enforce: 보안 없으면 통신 차단 Permissive: 비보안 노드도 허용

※ 위 환경변수 3가지는 노드를 실행할 때마다 매번 각 터미널에서 선언해야 함.

## 노드 실행

```
ros2 run demo_nodes_cpp talker --ros-args --enclave /talker_listener/talker
ros2 run demo_nodes_py listener --ros-args --enclave /talker_listener/listener
```

`--ros-args --enclave` 옵션: 노드가 `/talker_listener/talker` 보안 enclave를 사용하도록 지정  
enclave 이름은 키 생성 시 사용한 것과 정확히 일치해야 함