Week_11

5장 Component

- ROS에서 Nodelets 패키지는 단일 프로세스에서 복수의 노드를 동작시킬 때
  zero-copy를 제공하여 메모리 공간을 절약할 수 있고

- ROS2에서는 이러한 문제를 IPC(Intra-Process Communication)을 사용하여 해결할 수 있다.
  (IPC는 rclcpp에서만 지원한다)

- 일반적으로, Node는 실행 파일로 컴파일되지만,
- Nodelets는 이와 다르게 공유 라이브러리(Shared library)로 컴파일되어 실행중인 컨테이너 프로세스(Container process)에 로드된다

- 컴파일 과정에서 링킹은 링커가 **여러 오브젝트 파일**을 **실행 파일로 생성**하는 것이고,
  이러한 링킹의 종류는
  - **정적 링킹**과(Static linking)
  - **동적 링킹**이(Dynamic linking) 있다.

- 정적 링킹은 (Static linking)
  라이브러리를 해당 **실행 파일에 저장**하는 것으로
  - 실행 파일마다 라이브러리를 저장하는 것이고
  - 특정 라이브러리가 수정되면 전체 프로그램을 다시 컴파일해야 한다는 단점을 갖는다.

- 동적 링킹은 (Dynamic linking)
  런타임 시간동안 **메모리에 공유 라이브러리가 있으면** → 해당 라이브러리에 접근하여 사용하고
  메모리 공유 라이브러리가 **없으면** → 해당 라이브러리를 **메모리에 올려 사용**하는 것을 말한다.

  - 공유 라이브러리를 여러 실행 파일이 사용하기 때문에
    - **메모리 관리가 효율적**이고
    - **라이브러리 유지 보수**가 편하지만,

    - 공유 라이브러리의 **주소를 찾고**, **접근**해야 하기 때문에 **성능이 떨어진다는 단점**이 있다.

5.2 동적 로딩과 ROS2 Component

- 동적 링킹(Dynamic linking)은 **프로그램을 실행했을때**, 공유 라이브러리를 메모리에서 찾는 것이지만
- 동적 로딩(Dynamic loading)은 **런타임**에 프로그램이 공유 라이브러리 사용을 결정하는 것이다.

- ROS1에서는 동적 로딩을(Dynamic loading)
  - class_loader
  - pluginlib

    패키지를 통해 지원한다.

- ROS2에서는 동적 로딩을(Dynamic loading)
  - class_loader, pluginlib을 통해서도 지원하지만,

    차이점은
  - pluginlib 기반으로 동작했던 Nodelets가, class_loader 기반의 Components를 제공한다는 점이다.

  - Components의 특징은
    - **main함수를 통해서** 노드를 직접 실행할 수도 있고
    - main함수 **없이도** 런타임에 컨테이너로 **동적 로딩**(Dynamic loading)되어 실행될 수 있다.

      → Components는 rclcpp에서만 사용 가능하고
    - **rclcpp::Node 클래스**를 상속받은 자식 클래스이거나, 멤버 변수로 갖고있는 클래스를

      로드할 수 있기 때문에
      → publish, subscribe, server, client **기능 또한 사용할 수 있다**는 특징을 가진다.

5.3 데모 코드

TalkerComponent

talker_component.hpp

```
#ifndef COMPOSITION__TALKER_COMPONENT_HPP_
#define COMPOSITION__TALKER_COMPONENT_HPP_

#include "composition/visivility_control.h"
#include "rclcpp/rclcpp.hpp"
#include "std_msgs/msg/string.hpp"

namespace composition
{

class Talker : public rclcpp::Node
{

public:
  COMPOSITION_PUBLIC
  explicit Talker(const rclcpp::NodeOptions & options);

protected:
  void on_timer();

private:
  size_t count;
  rclcpp::Publisher<std_msgs::msg::String>::SharedPtr pub_;
  rclcpp::TimerBase::SharedPtr timer_;
};

} // namespace composition

#endif //COMPOSITION__TALKER_COMPONENT_HPP_
```

- visibility_control에서의 visibility는 특정한 함수나 변수(symbol)가 **다른 파일로 연결될 수 있는지** 여부를 나타낸다.
- 파일 포함(#include)과는 별개의 개념이다.

talker_component.cpp 일부

```
#include "rclcpp_components/register_node_macro.hpp"

// Register the component with class_loader.
// This acts as a sort of entry point,
//allowing the component to be discoverable when its library
// is being loaded into a running porcess.
RCLCPP_COMPONENTS_REGISTER_NODE(composition::Talker)
```

- 노드를 컴포넌트로 사용하려면, rclcpp::Node를 상속받은

  namespace composition

  {

  } // namespace composition

  문 밖에

```
#include "rclcpp_components/register_node_macro.hpp"

// Register the component with class_loader.
// This acts as a sort of entry point,
//allowing the component to be discoverable when its library
// is being loaded into a running porcess.
RCLCPP_COMPONENTS_REGISTER_NODE(composition::Talker)
```

매크로를 이용하여, 해당 클래스를 별도로 등록해 주어야

동적 로딩(Dynamic loading)시 컨테이너가 해당 라이브러리의 위치를 확인할 수 있다.

CmakeLists파일에서는

rclcpp_components의 의존성을 명시해 준다.

```
find_package(rclcpp_components REQUIRED)
```

작성한 노드를 add_library 명령어를 이용하여 공유 라이브러리로 만든다

```
add_library(talker_component SHARED
  src/talker_component.cpp)
```

rclcpp_components_register_nodes 명령어를 이용하여 노드에서 컴포넌트를 사용할 수 있도록 등록한다.

```
rclcpp_components_register_nodes(talker_component "composition::Talker")
```

- 노드에서 컴포넌트를 사용할 수 있도록 등록하면, Dynamic loading을 사용할 수 있게 된다.

---

rclcpp::Node를 상속받지 않는 클래스여도 다음 조건이 충족되면 component로 사용할 수 있다.

1. rclcpp::Node를 멤버 변수로 갖는다
2. 클래스 생성자의 매개변수로 오직 rclcpp::NodeOptions를 가진다
3. 컨테이너로 노드의 인터페이스를 전달해줄 함수를 가진다.

- 컨테이너는 컴포넌트(Component)들을 동적으로 로드하고 관리하는 ROS 시스템의 일부이다.
- Node의 직접 상속을 받지 않더라도, 멤버 변수로부터만 매개변수를 갖고, 노드 인터페이스를 전해주면 component로 사용 가능하

5.4 실행 방법

- 컴포넌트를 사용하는 것은 세 가지 방법이 있다.
  1. Generic container process의 service 통신을 이용해서 컴포넌트를 등록하는 방법
  2. main함수에 executors를 선언해서 컴포넌트를 등록하는 방법
  3. Launch를 이용하여 여러 개의 컴포넌트를 한 번에 등록하는 방법

5.4.1 Generic container process

- **런타임**중에 **컨테이너에 컴포넌트를 등록**하기 위해서는 **service 통신**을 이용해야 한다.

- CLI에서 다음 명령어로 워크스페이스(workspace)에서 사용 가능한 컴포넌트를 찾아보자

```
$ ros2 component types
```

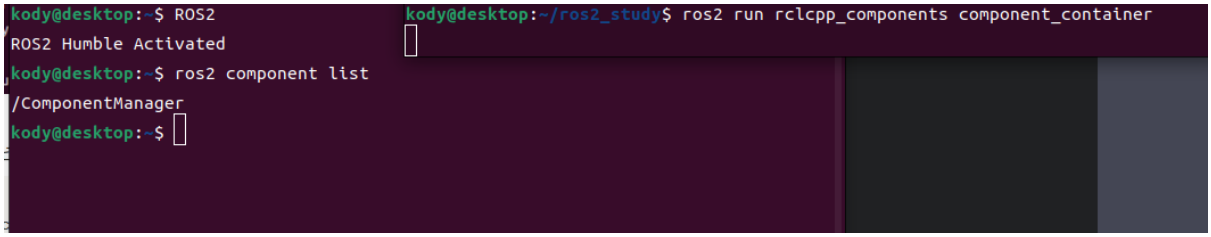
```
kody@desktop:~/ros2_study$ ROS2
ROS2 Humble Activated
kody@desktop:~/ros2_study$ ros2study
ros2_study workspace is activated
kody@desktop:~/ros2_study$ ros2 component types
composition
  composition::Talker
  composition::Listener
  composition::NodeLikeListener
  composition::Server
  composition::Client
action_tutorials_cpp
  action_tutorials_cpp::FibonacciActionClient
  action_tutorials_cpp::FibonacciActionServer
nav2_controller
  nav2_controller::ControllerServer
depthimage_to_laserscan
  depthimage_to_laserscan::DepthImageToLaserScanROS
tf2_ros
  tf2_ros::StaticTransformBroadcasterNode
logging_demo
  logging_demo::LoggerConfig
  logging_demo::LoggerUsage
nav2_behaviors
  behavior_server::BehaviorServer
joy
  joy::Joy
nav2_lifecycle_manager
  nav2_lifecycle_manager::LifecycleManager
demo_nodes_cpp_native
  demo_nodes_cpp_native::Talker
examples_rclcpp_minimal_subscriber
  WaitSetSubscriber
  StaticWaitSetSubscriber
  TimeTriggeredWaitSetSubscriber
demo_nodes_cpp
  demo_nodes_cpp::OneOffTimerNode
  demo_nodes_cpp::ReuseTimerNode
  demo_nodes_cpp::ServerNode
  demo_nodes_cpp::ClientNode
  demo_nodes_cpp::ListParameters
  demo_nodes_cpp::ParameterBlackboard
  demo_nodes_cpp::SetAndGetParameters
  demo_nodes_cpp::ParameterEventsAsyncNode
  demo_nodes_cpp::EvenParameterNode
  demo_nodes_cpp::ContentFilteringPublisher
```

다음 명령어로

1. 컨테이너를 실행하고
2. 컴포넌트 리스트를 확인한다

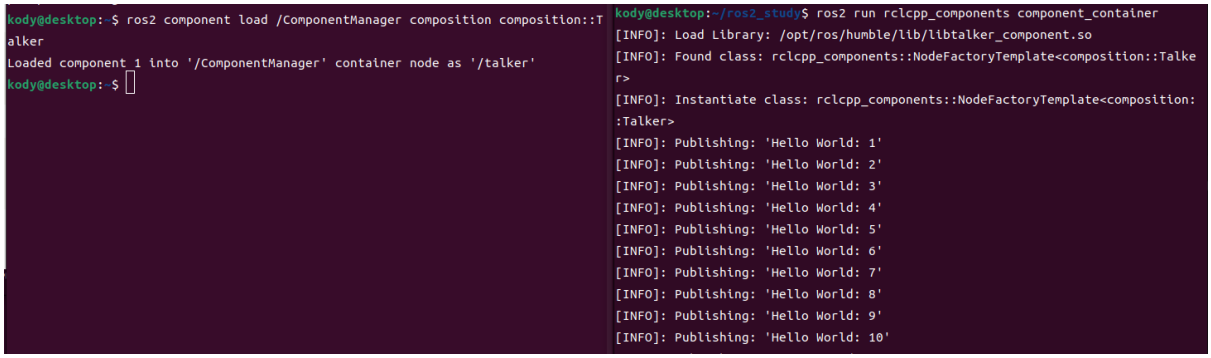
```
$ ros2 run rclcpp_compoenents component_container
```

```
$ ros2 component list
```



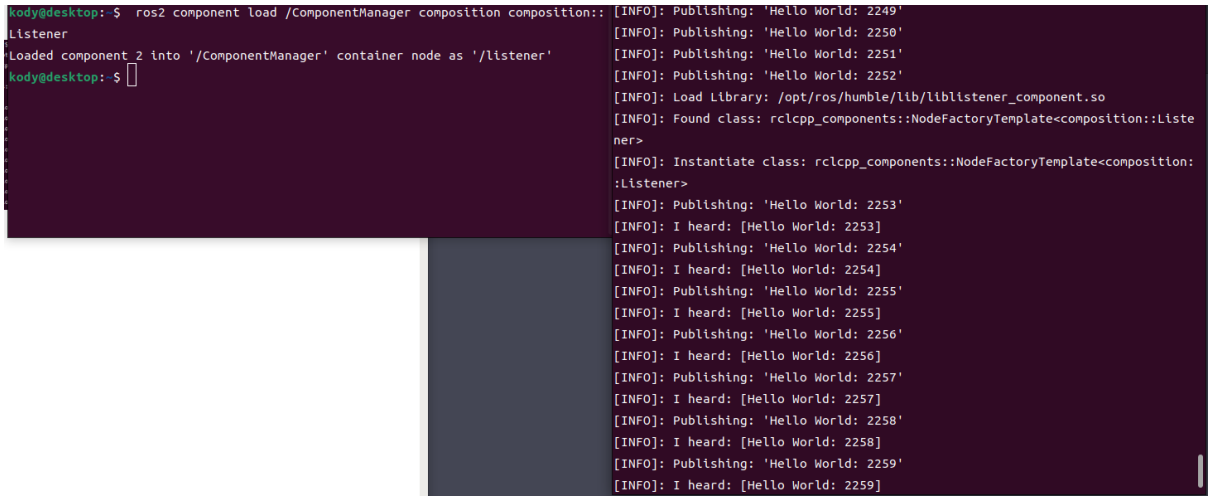
- talker 컴포넌트를 실행 중인 컨테이너에 적재해 본다.

컴포넌트 컨테이너에서 talker가 실행되는 것을 확인할 수 있다.

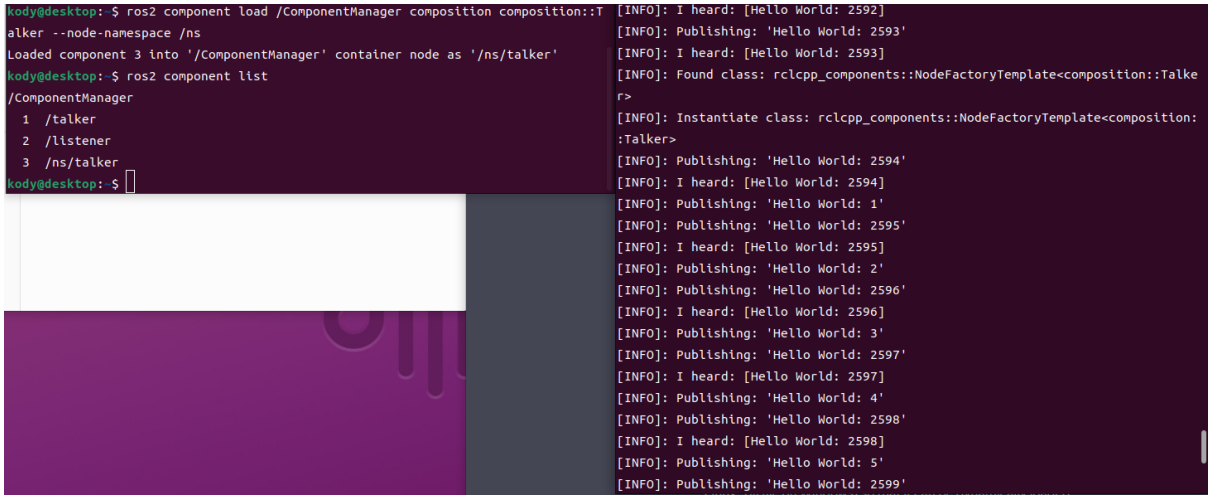


- 컨테이너에 listener 컴포넌트를 등록하면

  컨테이너에서 talker과 listener의 로그를 확인할 수 있다.



- 별도의 네임스페이스를 붙혀 talker 컴포넌트를 적재한다.



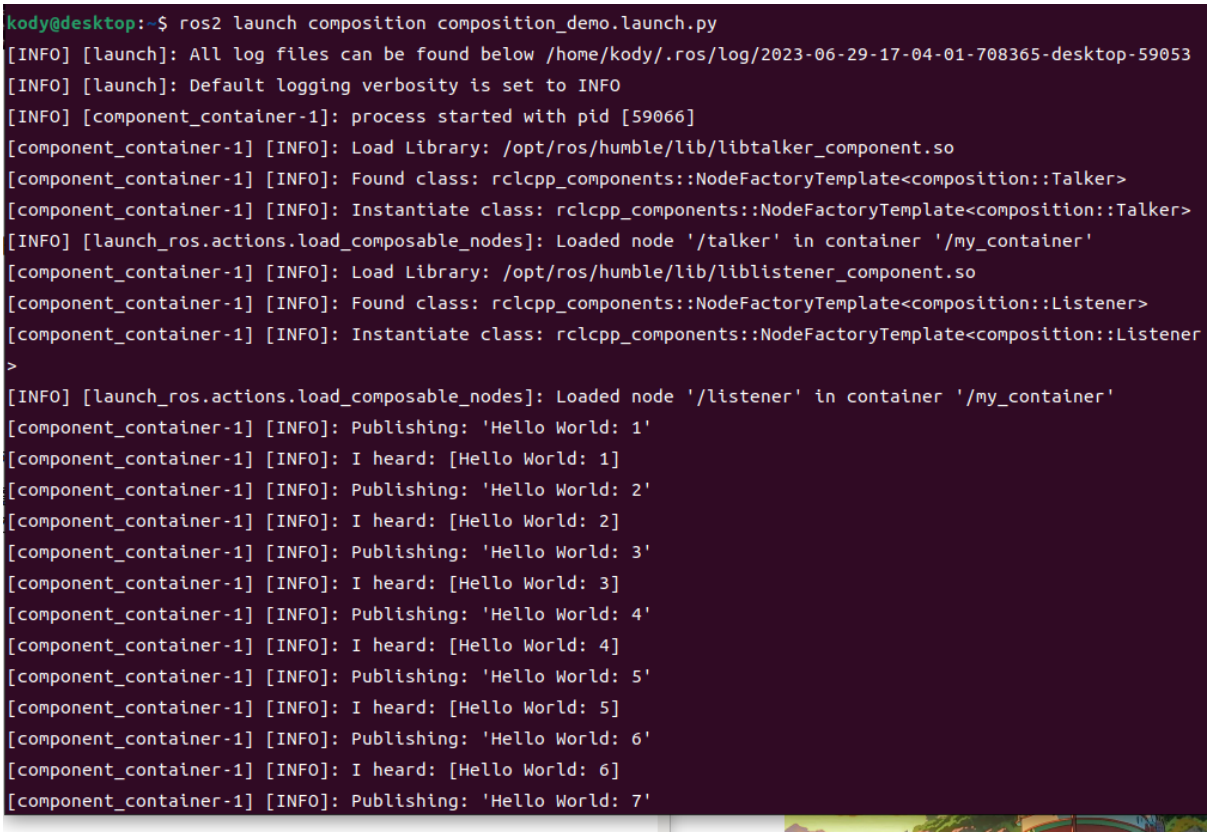
5.4.2 Custom Executable

- main함수를 작성하여 컴포넌트를 하나의 executor에 등록하는 방법이다.
  컴파일 타임에 공유 라이브러리들이 동적 링킹된다.

5.4.3 Launch

- 복수 개의 노드를 실행시킬 때 사용하는 launch파일을 이용해서도 컴포넌트를 사용 가능하다.
- launch_ros **모듈**의 **ComposableNodeContainer** 클래스를 이용해서 컨테이너 이름과 네임스페이스를 지정할 수 있고 **Composab**

- launch파일 실행



- launch파일을 실행해서 talker노드와 listener 노드를 실행했다.

6장 RQt plugin

6.1 RQt

- RQt는 ROS의 Qt로 종합GUI 툴박스이다.
- 파이썬 기반 RQt 플러그인을 신규로 작성해본다.

Rqt관련 패키지

- rqt패키지
  - 메타 패키지로, rqt_gui / rqt_gui_cpp / rqt_gui_py / rqt_py_common패키지를 포함한다.
- rqt_gui
  - 여러 RQt유젯을 단일 창에 도킹할 수 있게 한다.
- rqt_gui_cpp
  - C++ 클라이언트 라이브러리로 제작할 수 있는 RQt GUI 플러그인 API를 제공하는 패키지이다.
- rqt_gui_py
  - 파이썬 클라이언트 라이브러리로 제작할 수 있는 RQt GUI 플러그인 API를 제공하는 패키지이다.
- 등등의 패키지가 있다.

6.4 RQt 플러그인 작성 순서

특징점으로는

- 파이썬 언어로 작성되지만, 빌드를 ament_cmake로 해야 한다는 것이다.

다음은 플러그인 작성 순서이다.

1. RQt 플러그인 패키지 생성

```
kody@desktop:~/ros2_study/src$ ROS2
ROS2 Humble Activated
kody@desktop:~/ros2_study/src$ ros2study
ros2_study workspace is activated
kody@desktop:~/ros2_study/src$ ros2 pkg create my_first_rqt_plugin_pkg --build-t
ype ament_cmake --dependencies rclpy rqt_gui rqt_gui_py python_qt_binding
going to create a new package
package name: my_first_rqt_plugin_pkg
destination directory: /home/kody/ros2_study/src
package format: 3
version: 0.0.0
description: TODO: Package description
maintainer: ['kody <pg01198@naver.com>']
licenses: ['TODO: License declaration']
build type: ament_cmake
dependencies: ['rclpy', 'rqt_gui', 'rqt_gui_py', 'python_qt_binding']
creating folder ./my_first_rqt_plugin_pkg
creating ./my_first_rqt_plugin_pkg/package.xml
creating source and include folder
creating folder ./my_first_rqt_plugin_pkg/src
```

2. 패키지 설정 파일 수정

```
my_first_rqt_plugin_pkg/package.xml
```

3. 플러그인 파일 생성

```
my_first_rqt_plugin_pkg/package.xml
```

4. 빌드 설정 파일 수정

```
my_first_rqt_plugin_pkg/CMakeList.txt
```

5. 스크립트 폴더 및 파일 생성

```
my_first_rqt_plugin_pkg/scripts/my_first_rqt_plugin_pkg
```

6. 리소스 폴더 및 UI 파일 생성

```
my_first_rqt_plugin_pkg/resource/my_first_rqt_plugin_pkg.ui
```

7. 소스 폴더 및 파일 생성

```
__init__.py
example.py
examples_widget.py
examples_xxxxxxx.py
```

8. 런치 폴더 및 런치 파일 생성

```
my_first_rqt_plugin_pkg/launch/rqt_plugin.launch.py
```

6.7 RQt 예제 살펴보기

1. git clone한다.

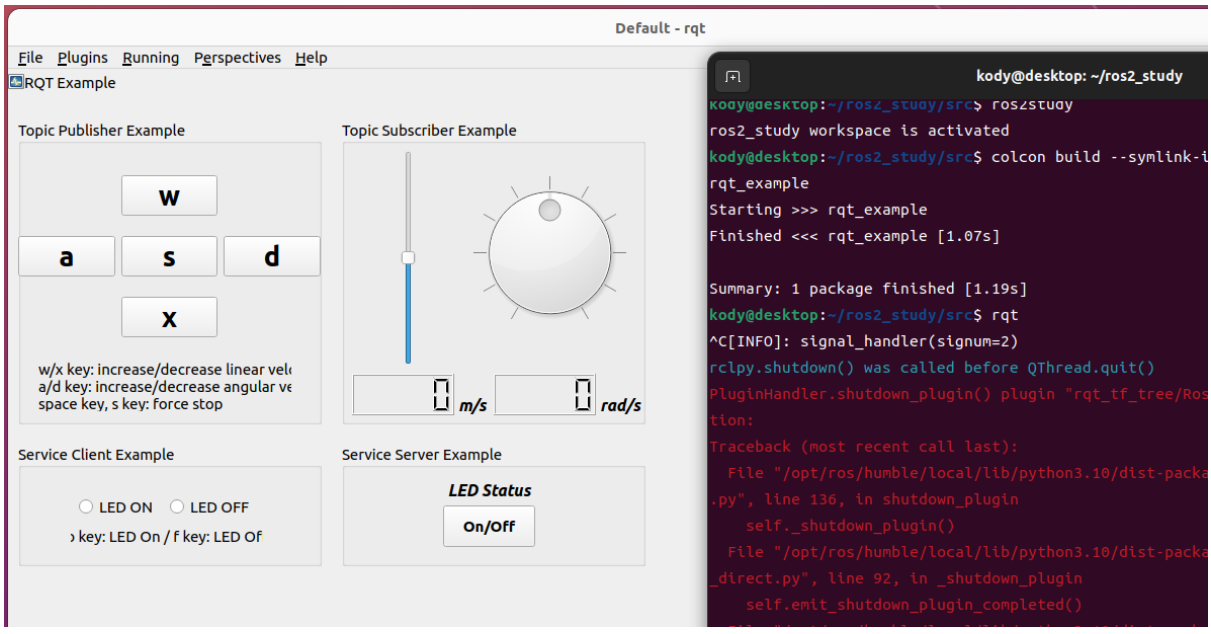
```
git clone https://github.com/robotpilot/ros2-seminar-examples.git
```

빌드후 실행

```
$ ros2 run rqt_example rqt_example
```

실행시

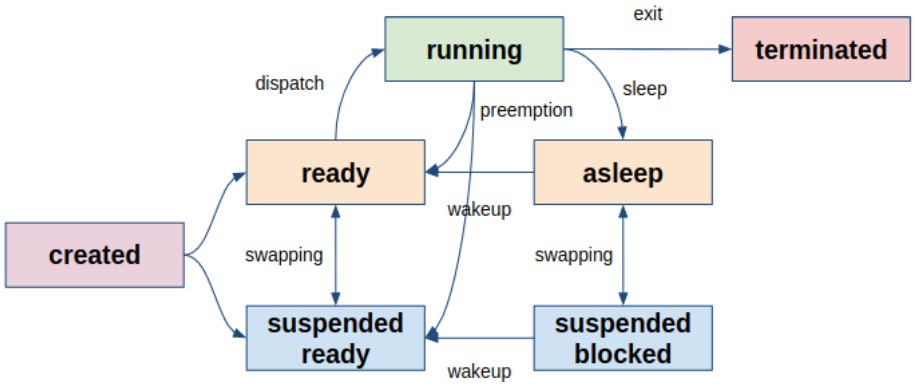
qt_gui_main() found no plugin matching에러 → rqt —force-discover 명령어로 실행



7장 Lifecycle

7.1 노드 관리

- 운영체제는 복수 개의 프로세스를 관리하기 위해서 프로세스의 상태를 정의하고, 상태의 전환을 조율하는 역할을 수행한다.

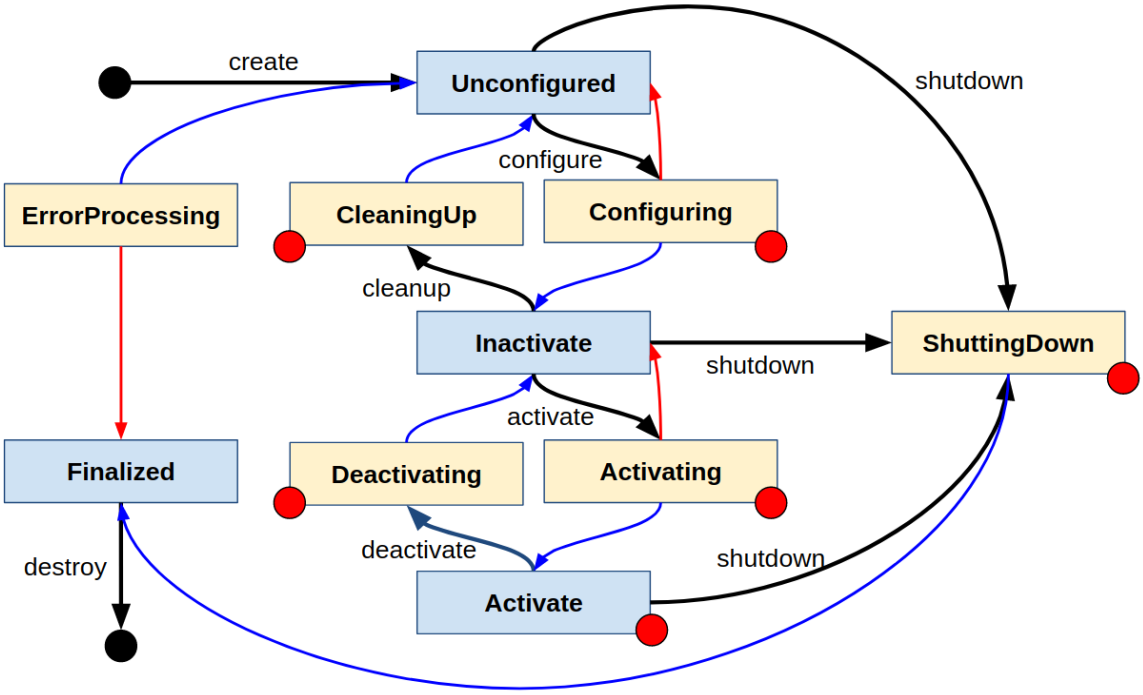


- 이와 같이 ROS2에서는 노드(Node)의 상태를 확인, 관리하기 위한 인터페이스인 Lifecycle을 API 형태로 제공한다.
  - Lifecycle은 인터페이스로
    - 주요 상태(Primary states)
    - 전환 상태(Transition states)
    - 전환(Transition)을 제공한다

7.2 Lifecycle

- 상태와 상태전환

  ROS2 Lifecycle



- 파란색 박스는 주요 상태를 의미하고
- 노란색은 전환 상태를 이야기한다.

- 검은 선은 전환을 표기한 것이고,
- 파란 화살표는 전환 상태가 성공일 때를 의미하고
- 빨간 화살표는 전환 상태가 실패일 때를 의미한다.

- Lifecycle에는 4개의 주요 상태가 정의되어 있다.
  - Uncinfigured
    - 노드가 생성된 직후 / 에러 발생 이후 다시 조정이 필요한 상태
  - Inactive
    - 노드가 동작을 수행하지 않는 상태로, 파라미터 등록, 토픽 퍼블리시, 서브스크라이브 추가 삭제등을 할 수 있다.
  - Active
    - 동작을 수행중인 상태이다.
  - Finalized
    - 노드가 메모리에서 해체되기 전의 상태이다. 디버깅이나 내부 검사등을 진행할 수 있다.

- Lifecycle에는 6개의 전환 상태가 정의되어 있다.
  - Configuring
    - 노드를 구성하기 위한 전환을 수행한다
  - CleaningUp
    - 노드를 초기 상태로 만드는 전환을 수행한다

    - Activating
        - 동작 수행 상태로 만드는 전환을 수행한다
    - Deactivating
        - 동작 수행 전으로 되돌리는 전환을 수행한다
    - ShuttingDown
        - 노드를 파괴하는 전환을 수행한다
    - ErrorProcessing
        - 사용자 코드가 동작하는 상태에서 발생하는 에러를 해결하는 과정을 수행한다.

- 7개의 전환이 정의되어 있다.
    - Create
    - Configure
    - Cleanup
    - Activate
    - Deactivate
    - Shutdown
    - Destroy

8장 Security

→일단은 기능 구현이 먼저라고 생각해서 패스