RRM

ROS komunikácia
Node, Topic

Michal Dobiš
michal.dobis@stuba.sk

Marek Čornák
marek.cornak@stuba.sk

Jakub Ivan
jakub.ivan@stuba.sk

Obsah

1. Úvod do C++ - dedičnosť
2. ROS Node a Topic
3. Balík rrm_sim
4. Publisher
5. Subscriber
6. std::shared_ptr
7. Samostatná úloha

Úvod do C++ - Dedičnosť

- základná vlastnosť OOP
- Odvodená trieda dedí vlastnosti (atribúty a metódy) z existujúcej materskej triedy
- Zmysel: lepšia využiteľnosť kódu, zníženie redundancie, štrukturovanie a hierarchia
- Prístupové špecifikátory (Access Specifiers)
  - Public
  - Private
  - Protected

Doplnkovy zdroj: https://www.geeksforgeeks.org/inheritance-in-c/

Úvod do C++ - Dedičnosť - base class

```
#include <iostream>

class Animal {
private:
	int age; // Private member

public:
	Animal(int a) : age(a) {
		std::cout << "Animal constructor called. Age initialized to " << age << "." << std::endl;
	}

	void eat() {
		std::cout << "This animal is eating." << std::endl;
	}

	int getAge() const{
		return age;
	}
};
```

Úvod do C++ - Dedičnosť - inherited class

```
class Dog :  public Animal {
public:
        Dog(int a) : Animal(a) {
        std::cout << "Dog constructor called." << std::endl;
        }

        void bark() {
        std::cout << "The dog is barking." << std::endl;
        }
};
```

public - Dedenie ako public
private - všetky public funkcie z base class budú private

Inicializácia **base constructor**

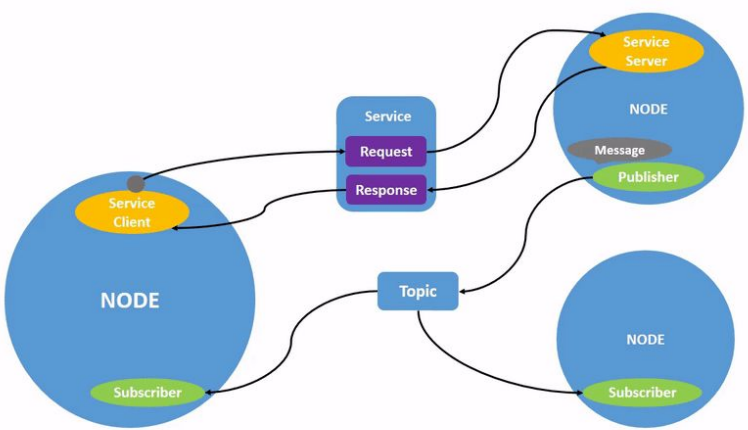
Implementovanie novej funkcie dostupnej iba v triede Dog

Úvod do C++ - Dedičnosť

```
int main() {
        Dog myDog(5);
        myDog.eat();
        myDog.bark();
        return 0;
}
```

```
Animal constructor called. Age initialized to 5.
Dog constructor called.
This animal is eating.
The dog is barking.
```

ROS Node

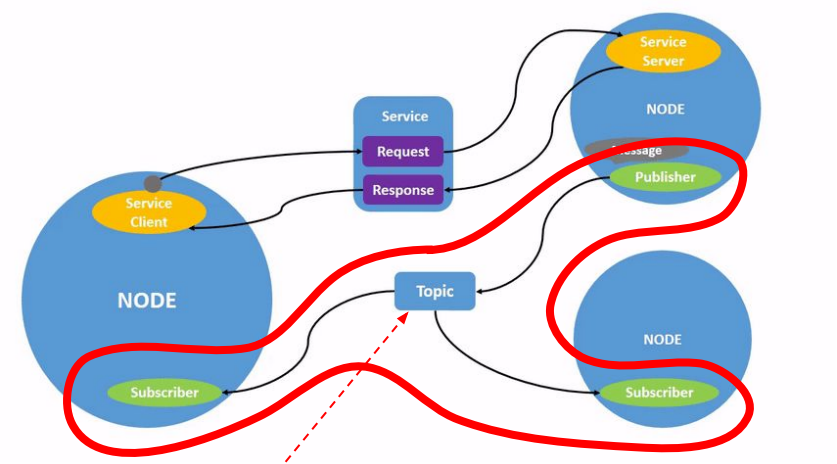


- Spustiteľné programy (executables)
  - Binárny súbor z C++
  - Python script
- Jedno účelové
  - Riadenie
  - snímače
  - rôzne algoritmy
- Komplexný robotický systém pozostáva z viacerých node.
- Spustenie

ros2 run <package_name> <executable_name>

- Zdroj

ROS Node - Komunikácia Topics



*.msg

```
int number
double width
string  descripion
etc...
```

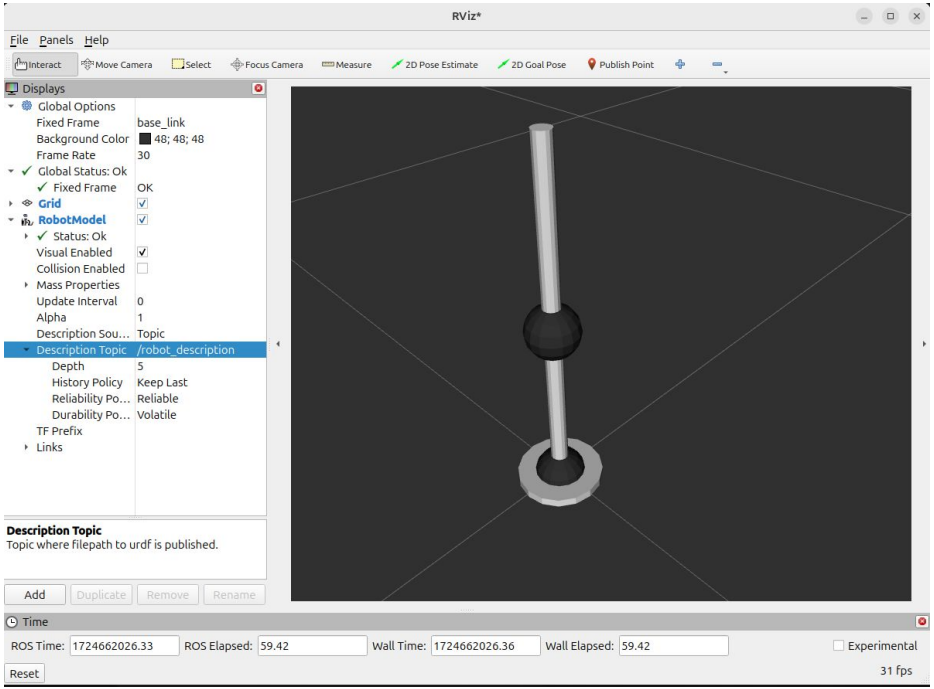
- **Topics**
  - Asynchrónna (jednosmerná) komunikácia
  - Princíp publish – subscribe
  - Dátový obsah určený štruktúrou správy - message (*.msg)
- **Publisher**
  - Odosiela dáta
    - periodicky napr. dáta z enkóderov
    - udalosť napr. zmena stavu
  - Môže mať viacero odberateľov
- **Subscriber**
  - Prijíma a spracováva dáta z topicu
  - napr. odhad pozicie z odometrie
- Zdroj

Balík rrm_sim

Spustenie:
ros2 launch rrm_sim rrm_sim.launch.xml

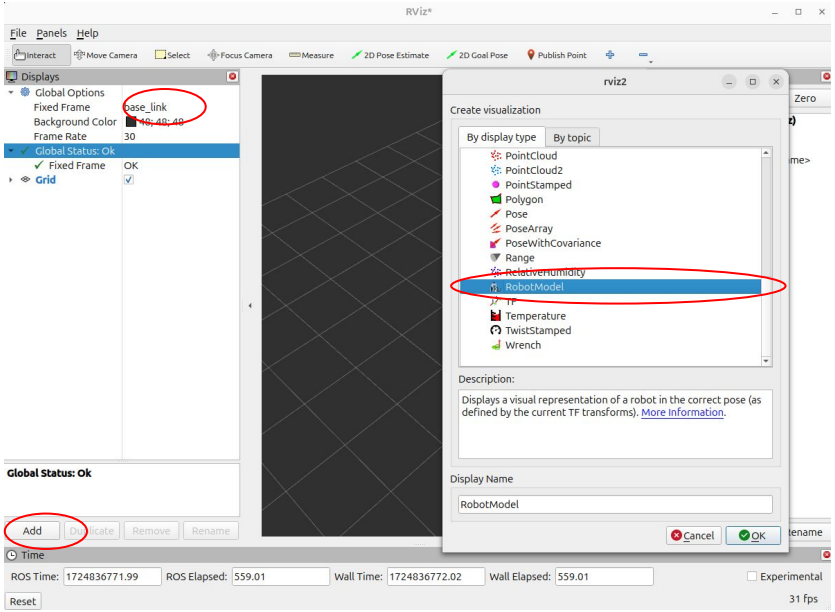
Pozostáva z:

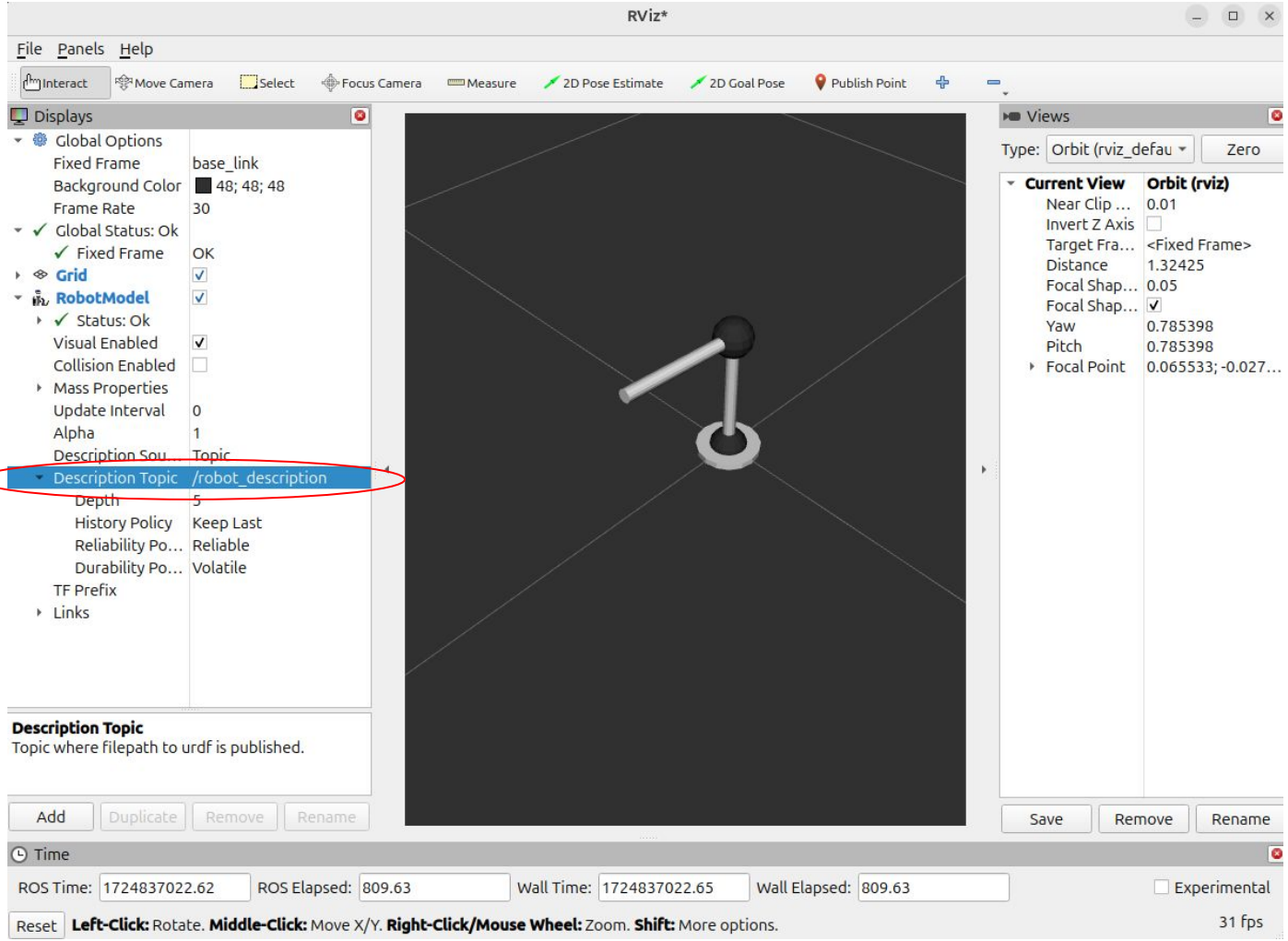
- **rrm_sim** - Jednoduchá kinematická simulácia robota (nezohľadňuje dynamiku a zrýchlenie robota je nekonečné)
- **robot_state_publisher** - počíta priamu kinematiku
- **rviz** - vizualizácia



Nastavenie vizualizácie robota - RViz

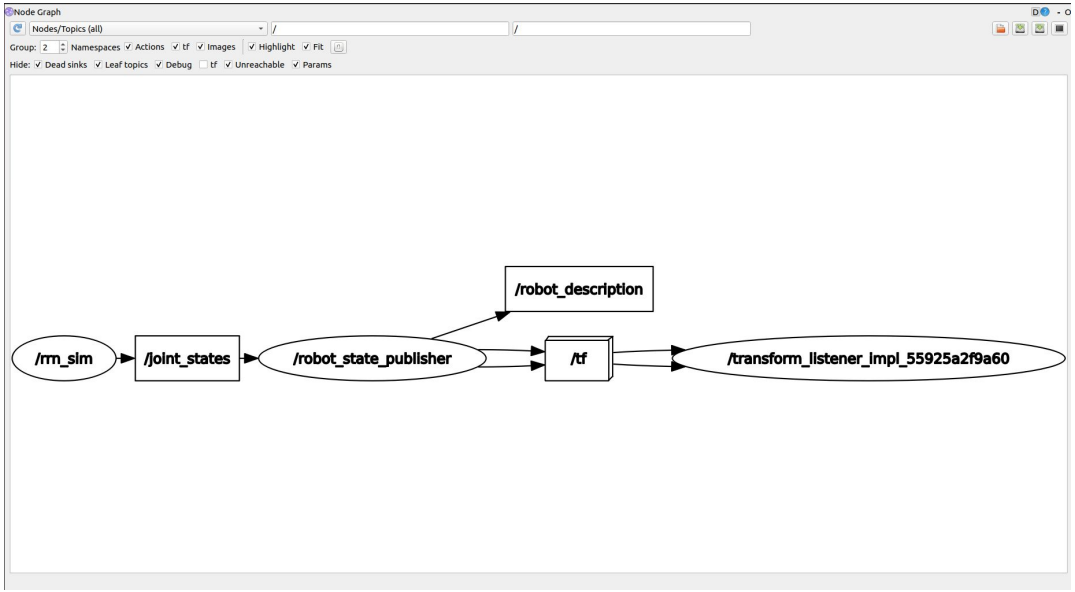
- Otvorí sa okno RViz
- V Global options vlavo zmenit **Fixed Frame** z map na **base_link**
- V RobotModel pridat Description Topic na **/robot_description**




RViz*
File Panels Help
Interact
Move Camera
Select
Focus Camera
Measure
2D Pose Estimate
2D Goal Pose
Publish Point
Displays
Global Options
Fixed Frame base_link
Background Color 48; 48; 48
Frame Rate 30
Global Status: Ok
Fixed Frame OK
Grid
RobotModel
Status: Ok
Visual Enabled
Collision Enabled
Mass Properties
Update Interval 0
Alpha 1
Description Sou... Topic
Description Topic /robot_description
Depth 5
History Policy Keep Last
Reliability Po... Reliable
Durability Po... Volatile
TF Prefix
Links
Description Topic
Topic where filepath to urdf is published.
Add
Duplicate
Remove
Rename
Views
Type: Orbit (rviz_defau
Zero
Current View Orbit (rviz)
Near Clip ... 0.01
Invert Z Axis
Target Fra... <Fixed Frame>
Distance 1.32425
Focal Shap... 0.05
Focal Shap...
Yaw 0.785398
Pitch 0.785398
Focal Point 0.065533; -0.027...
Save
Remove
Rename
Time
ROS Time: 1724837022.62
ROS Elapsed: 809.63
Wall Time: 1724837022.65
Wall Elapsed: 809.63
Experimental
Reset
Left-Click: Rotate. Middle-Click: Move X/Y. Right-Click/Mouse Wheel: Zoom. Shift: More options.
31 fps

rqt_graph

Príkaz: **rqt_graph**

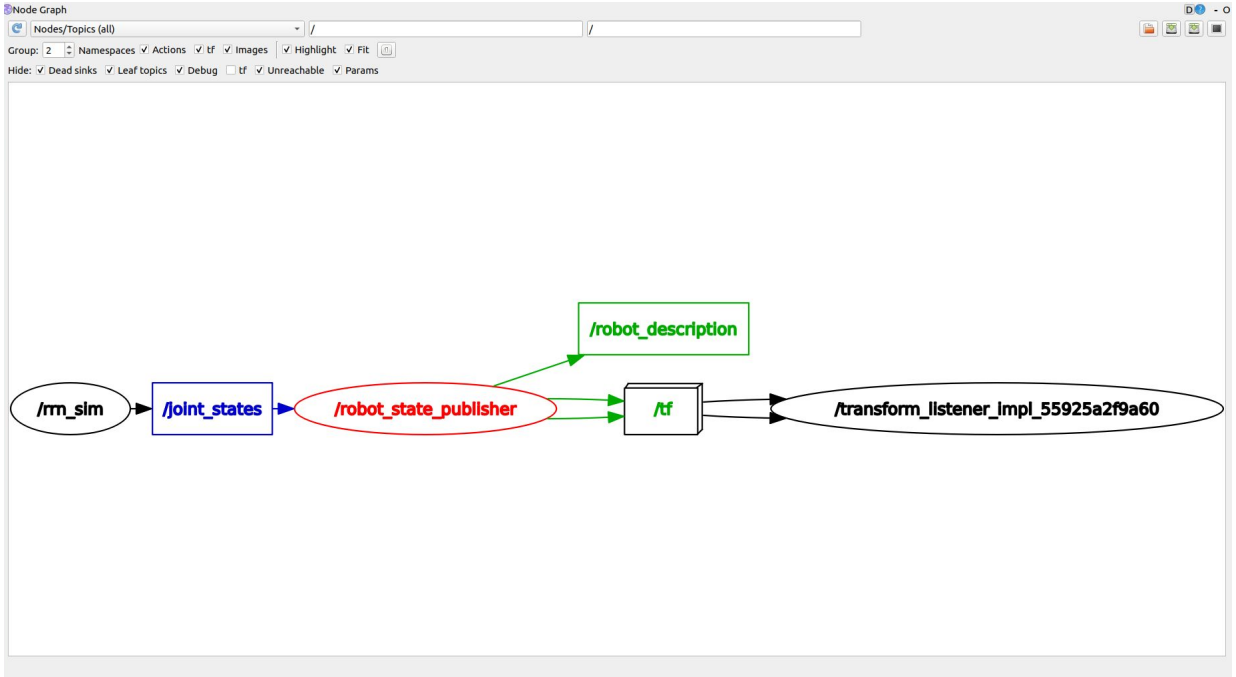


Tip: príkaz **rqt** pre gui monitorovanie štruktúry vášho projektu

Node robot_state_publisher

Počúva aktuálnu polohu robota z /joint_states

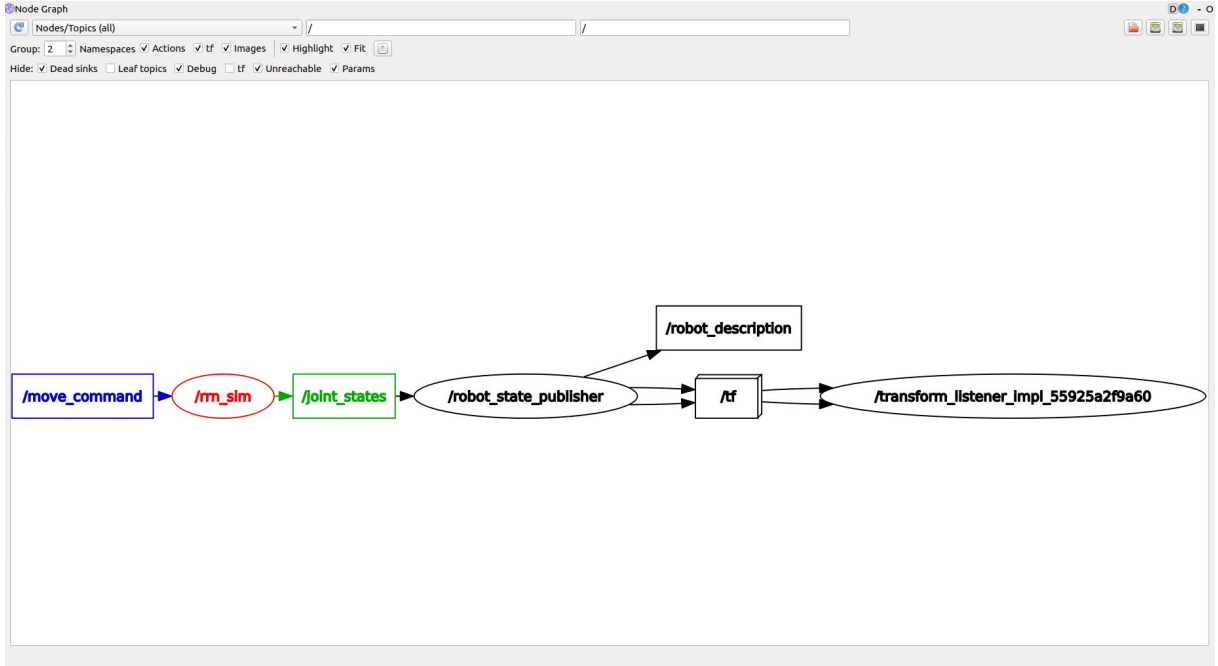
Vypočíta priamu kinematiku a aktuálny stav robota posiela na /tf



Simulácia rrm_sim

Počúva pohybový príkaz na /move_command

Posiela aktuálne natočenie kĺbov na /joint_states



Balík rrm_sim - robot_sim node

- publikuje polohu kĺbov na topic **joint_state** správu **sensor_msgs/JointState**
- Zistenie topicu zo zoznamu: ros2 topic list
- Vypisanie dát na topicu cez CLI: **ros2 topic echo /joint_state**

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/rrm_ws$ ros2 topic echo /joint_states
header:
  stamp:
    sec: 1724662828
    nanosec: 804548350
  frame_id: ''
name:
- joint_1
- joint_2
- joint_3
position:
- 0.0
- 0.0
- 0.0
velocity:
- 0.0
- 0.0
- 0.0
effort: []
---
```

Simulácia rrm_sim

Vyhľadanie node v CLI: ros2 node list

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/rrm_ws$ ros2 node list
/robot_sim
/robot_state_publisher
/rviz2
/transform_listener_impl_5efe6b1f0e60
```

Informácie o node: ros2 node info /<node_name>

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/rrm_ws$ ros2 node info /robot_sim
/robot_sim
  Subscribers:
    /move_command: rrm_msgs/msg/Command
    /parameter_events: rcl_interfaces/msg/ParameterEvent
  Publishers:
    /joint_states: sensor_msgs/msg/JointState
    /parameter_events: rcl_interfaces/msg/ParameterEvent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /move_command: rrm_msgs/srv/Command
    /robot_sim/describe_parameters: rcl_interfaces/srv/DescribeParameters
    /robot_sim/get_parameter_types: rcl_interfaces/srv/GetParameterTypes
    /robot_sim/get_parameters: rcl_interfaces/srv/GetParameters
    /robot_sim/get_type_description: type_description_interfaces/srv/GetTypeDescription
    /robot_sim/list_parameters: rcl_interfaces/srv/ListParameters
```

Balík rrm_sim

- robot je možné ovládať publikovaným na topic **/move_command**
- Zistenie typu správy: **ros2 topic info /move_command**

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/rrm_ws$ ros2 topic info /move_command
Type: rrm_msgs/msg/Command
Publisher count: 0
Subscription count: 1
```

- Zistenie štruktúry správy: **ros2 interface show rrm_msgs/msg/Command**

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/rrm_ws$ ros2 interface show rrm_msgs/msg/Command
int32 joint_id
float64 position
```

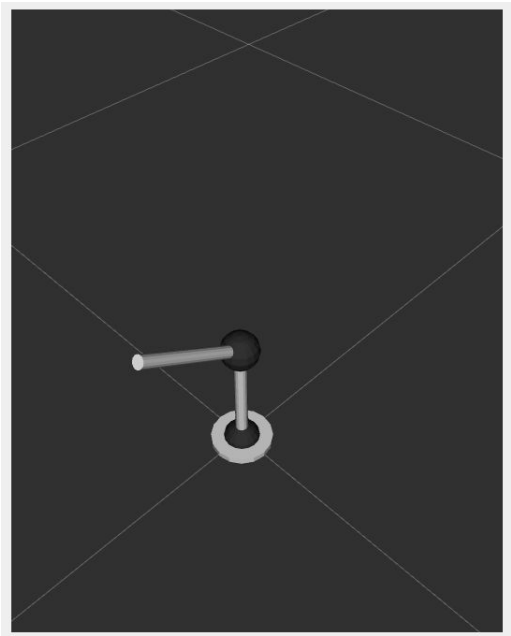
Balík rrm_sim

- Publikovanie správy na topic /move_command:

ros2 topic pub /move_command rrm_msgs/msg/Command "{joint_id: 2, position: 1.0}" -1

- Overenie pohybu pomocou CLI:
  ros2 topic echo /joint_states

```
marek@marek-ASUS-TUF-Gaming-A15-FA507NV-FA507NV:~/rrm_ws$ ros2 topic echo /joint_states
header:
  stamp:
    sec: 1724835935
    nanosec: 776764426
  frame_id: ''
name:
- joint_1
- joint_2
- joint_3
position:
- 0.0
- 0.0
- 1.0
velocity:
- 0.0
- 0.0
- 0.0
effort: []
---
```



ROS Node - implementácia

- Pre prístup k ROS rozhraniu (topics, services, atď…) je potrebné vytvoriť triedu ktorá dedí od triedy **rclcpp::Node**
- V main funkcii je potrebné túto triedu inštancovať
- Vytvorenie noveho pkg pre blok 1:

ros2 pkg create block1_<priezvisko> --build-type ament_cmake --dependencies rclcpp **rrm_msgs** --node-name teleop_node

ROS Node - príklad

```
#include "rclcpp/rclcpp.hpp"
class Teleop : public rclcpp::Node
{
  public:
   Teleop() : Node("Teleop")
{
   RCLCPP_INFO(this->get_logger(), "Teleop initialized");
   }
   void move(int joint_id, double position) {
       // TODO tu neskor napiseme ROS publisher
   }
};
int main(int argc, char ** argv)
{
rclcpp::init(argc, argv);
Teleop robot;
return 0;
}
```

ROS Node - publisher implementácia c++

Pridanie header file pre message interface:

```
#include "rrm_msgs/msg/command.hpp"
```

Deklarácia premennej publisher v triede Teleop:

```
private:
rclcpp::Publisher<rrm_msgs::msg::Command>::SharedPtr publisher_;
```

Inicializácia publishera v konštruktore triedy Teleop:

```
publisher_ = this->create_publisher<rrm_msgs::msg::Command>("move_command", 10);
```

Vyplnenie správy dátami podľa jej typu (vo vašej funkcii *move*):

```
rrm_msgs::msg::Command message;
message.joint_id = joint_id;
message.position = position;
```

Publikovanie správy na topic /move_command

```
publisher_->publish(message);
```

Odkaz na samoštúdium:
https://docs.ros.org/en/jazzy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

Vytvorenie druhej node so subscriberom

- V našom balíku budeme mať viacero nodes a k nim prisluchajúce zdroj. súbory
- Balík bude mať viacero závislostí
- Musíme vytvoriť nový executable target a prilinkovať potrebné závislosti
- **Package.xml:**

```
<depend>rclcpp</depend>
<depend>rrm_msgs</depend>
<depend>sensor_msgs</depend>          ⇐ (rozhranie pre "ROS-native" správy)
```

Ako bude vyzerať **CMakeLists.txt**:

- Pridanie ROS dependencies

```
find_package(ament_cmake REQUIRED)
find_package(rclcpp REQUIRED)
find_package(rrm_msgs REQUIRED)
find_package(sensor_msgs REQUIRED)
```

- Pridanie ďalšieho spusť. súboru/executable/node: **JointLogger**

```
add_executable(teleop_node src/teleop_node.cpp)
add_executable(logger_node src/logger_node.cpp)
```

Nezabudnite si vytvorit **logger_node.cpp**,

Vytvorenie druhej node

- Linkovanie pre novú node

```
target_include_directories(teleop_node PUBLIC
 $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
 $<INSTALL_INTERFACE:include/${PROJECT_NAME}>)
target_include_directories(logger_node PUBLIC
 $<BUILD_INTERFACE:${CMAKE_CURRENT_SOURCE_DIR}/include>
 $<INSTALL_INTERFACE:include/${PROJECT_NAME}>)
```

- Prilinkovanie závislostí do targetov/nodes

```
ament_target_dependencies(
 teleop_node
 "rclcpp"
 "rrm_msgs"
)
ament_target_dependencies(
 logger_node
 "rclcpp"
 "sensor_msgs"
)
```

- Inštalácia všetkých targetov

```
install(TARGETS teleop_node logger_node
        DESTINATION lib/${PROJECT_NAME})
```

ROS Subscriber Node

Definujeme triedu pre node JointLogger v subore logger_node.hpp

- deklarujeme premennú subscribera a obslužnú funkciu pre spracovane dát

```
#include "rclcpp/rclcpp.hpp"
#include "sensor_msgs/msg/joint_state.hpp"    //header file pre msg interface

class JointLogger : public rclcpp::Node
{
   public:
       JointLogger();
       void joint_states_callback(const sensor_msgs::msg::JointState::SharedPtr msg);
   private:
       rclcpp::Subscription<sensor_msgs::msg::JointState>::SharedPtr subscription_;
};
```

ROS Subscriber Node

Inicializácia subscribera v konštruktore triedy JointLogger:

```
JointLogger::JointLogger() : Node("joint_logger")
{
   subscription_ = this->create_subscription<sensor_msgs::msg::JointState>(
       "joint_states", 10, std::bind(&JointLogger::joint_states_callback, this, std::placeholders::_1));
}
```

Definovanie obslužnej callback funkcie pre subscribera:

```
   void JointLogger::joint_states_callback(const sensor_msgs::msg::JointState::SharedPtr msg)
   {
      RCLCPP_INFO(this->get_logger(), "First joint name: %s, position: %f", msg->name[0].c_str(),
                  msg->position[0]);
   }
```

ROS Subscriber Node

```
int main(int argc, char ** argv)
{
   rclcpp::init(argc, argv);
   std::shared_ptr<JointLogger> logger = std::make_shared<JointLogger>();
   //auto logger = std::make_shared<JointLogger>(); //alt. zapis

   rclcpp::spin(logger); //function that blocks the thread and allows the node to process callbacks
   rclcpp::shutdown();
   return 0;
}
```

Odkaz na samoštúdium:

https://docs.ros.org/en/jazzy/Tutorials/Beginner-Client-Libraries/Writing-A-Simple-Cpp-Publisher-And-Subscriber.html

Čo je to std::shared_ptr ?

- std::shared_ptr je smart pointer v C++.
- Automaticky spravuje dynamickú pamäť (počítanie referencii, aut. čistenie).
- Zdieľané vlastníctvo pamäte: Viaceré ukazovatele môžu ukazovať na rovnaký objekt.
- Referenčné počítanie: Uvoľní pamäť, keď už na objekt neukazuje žiadny shared_ptr.

Samoštúdium: https://www.geeksforgeeks.org/smart-pointers-cpp/

```cpp
#include <iostream>
#include <memory> // Potrebné pre smart pointery
class MyClass {
public:
   MyClass(int value) : value_(value) {
       std::cout << "MyClass instance created with value: " << value_ << std::endl;}
   ~MyClass() {
       std::cout << "MyClass instance destroyed" << std::endl;}
   void print_value() const {
       std::cout << "Value: " << value_ << std::endl;}
private:
   int value_;
};
int main() {
   // Vytvorenie shared pointera, ktorý vlastní objekt MyClass
   std::shared_ptr<MyClass> ptr1 = std::make_shared<MyClass>(10);
   // Vytvorenie druhého shared pointera, ktorý zdieľa rovnaký objekt ako ptr1
   std::shared_ptr<MyClass> ptr2 = ptr1;
   std::cout << "Reference count after creating ptr2: " << ptr1.use_count() << std::endl;
   // Použitie objektu cez ptr1
   ptr1->print_value();
   // Použitie objektu cez ptr2
   ptr2->print_value();
   // Po ukončení funkcie sa shared pointery ptr1 a ptr2 zničia a pamäť sa automaticky uvoľní
   std::cout << "Reference count before exiting: " << ptr1.use_count() << std::endl;
   return 0;
}
```