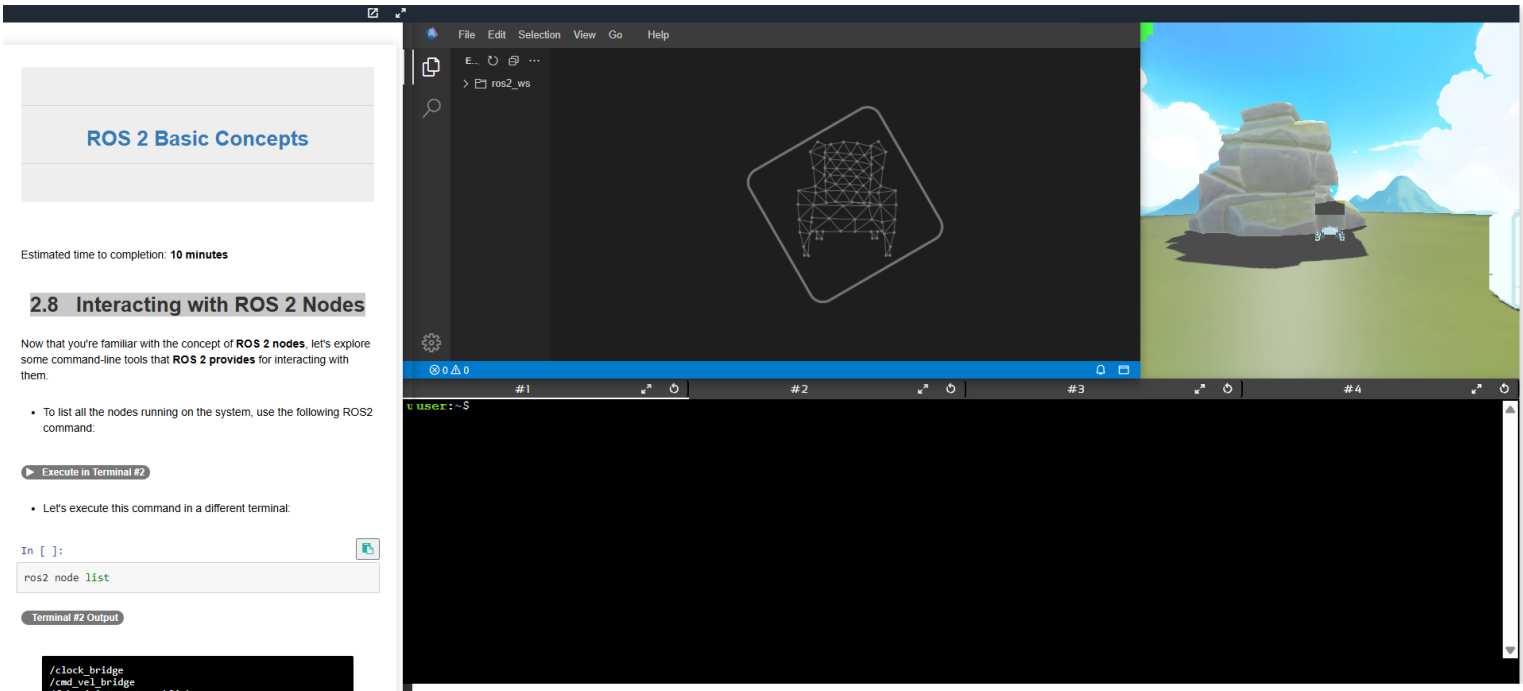

ROS 2 Basic Concepts

Estimated time to completion: **10 minutes**

2.8 Interacting with ROS 2 Nodes

Now that you're familiar with the concept of **ROS 2 nodes**, let's explore some command-line tools that **ROS 2 provides** for interacting with them.

- To list all the nodes running on the system, use the following ROS2 command:

Execute in Terminal #2

- Let's execute this command in a different terminal:

In [ ]:

```
ros2 node list
```

Terminal #2 Output

```
/clock_bridge
/cmd_vel_bridge
/fake_joint_state_publisher
/laser_bridge
/odom_bridge
/rgb_camera_bridge
/robot_state_publisher
```

- The nodes you're seeing right now are all related to the simulation.
- Follow the example below to run and visualize your own node.

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

```
source install/setup.bash
```

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable
```

Execute in Terminal #2

In [ ]:

```
ros2 node list
```

Terminal #2 Output

```
/clock_bridge
/cmd_vel_bridge
/fake_joint_state_publisher
/laser_bridge
/odom_bridge
/rgb_camera_bridge
/robot_state_publisher
```

- What happened?
- No new node appeared?
- Why?
- The problem is that our `heartbeat.py` `main()` method starts and stops too fast for us to see it appear in the `ros2 node list`.
- Let's change that!
- Let's add a while loop to keep the program active until we stop it manually:

heartbeat.py

In [ ]:

```
#!/usr/bin/env python
import rclpy
import time

def main(args=None):
    # initialize the ROS communication
    rclpy.init(args=args)

    i = 0
    max_i = 50
    while i < max_i:
        i += 1
        time_stamp = time.time()
        # print a message to the terminal
        print(str(i)+":Mars rover 1 is alive..."+str(time
        # Wait for 1 second
        time.sleep(1)

    # shutdown the ROS communication
    rclpy.shutdown()

if __name__ == '__main__':
    main() #call the main function
```

setup.py

In [ ]:

```
from setuptools import find_packages, setup

package_name = 'mars_rover_systems'

setup(
    name=package_name,
    version='0.0.0',
    packages=find_packages(exclude=['test']),
    data_files=[
        ('share/ament_index/resource_index/packages',
            ['resource/' + package_name]),
        ('share/' + package_name, ['package.xml']),
    ],
    install_requires=['setuptools'],
    zip_safe=True,
    maintainer='user',
    maintainer_email='user@todo.todo',
    description='TODO: Package description',
    license='TODO: License declaration',
    tests_require=['pytest'],
    entry_points={
        'console_scripts': [
            'heartbeat_executable = mars_rover_systems.he
        ],
    },
)
```

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

```
source install/setup.bash
colcon build
source install/setup.bash
```

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable
```

Execute in Terminal #2

In [ ]:

```
ros2 node list
```

Terminal #2 Output

```
/clock_bridge
/cmd_vel_bridge
/fake_joint_state_publisher
/laser_bridge
/odom_bridge
/rgb_camera_bridge
/robot_state_publisher
```

- **NO NODE APPEARS YET?!**
- **WHAT'S GOING ON?**
- There's a **SECOND ISSUE**.
- Unless we explicitly **START a ROS2 node**, ros2 won't start it for us automatically.
- Let's then introduce the python class `Node()`:

heartbeat.py

In [ ]:

```
#!/usr/bin/env python

import rclpy
import time
from rclpy.node import Node

def main(args=None):
    # initialize the ROS communication
    rclpy.init(args=args)

    # Create a node
    node = Node('mars_rover_1')

    i = 0
    max_i = 50
    while i < max_i:
        i += 1
        ros_time_stamp = node.get_clock().now()
        # print a message to the terminal
        node.get_logger().info(str(i)+":Mars rover 1 is a
        # Wait for 1 second
        time.sleep(1)


    # shutdown the ROS communication
    node.destroy_node()
    rclpy.shutdown()

if __name__ == '__main__':
    main()  # Call the main function
```

- Here are two main new elements to discuss:
- `node = Node('mars_rover_1')` : Here we initialize a **ROS2 node** with the name `mars_rover_1`. We can give it any name we want.
- `ros_time_stamp = node.get_clock().now()` : Instead of using the `time.time()` which retrieves the real time system clock time, the `node.get_clock().now()` gets the time used by **ALL ROS2 systems**, which don't necessarily be the same as the system clock. For example with simulations we can have a different clock from the real world. So from now on, time has to be retrieve in the ROS2 way.
- Also, in ROS2 the recommended way to print messages are **LOGS**. So from now on use the `node.get_logger().info()` instead of `print()`

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

```
source install/setup.bash
colcon build
source install/setup.bash
```

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable
```

Execute in Terminal #2

- Check that the `heartbeat_executable` is still running.

In [ ]:

```
ros2 node list
```

Terminal #2 Output

```
/clock_bridge
/cmd_vel_bridge
/fake_joint_state_publisher
/laser_bridge
/mars_rover_1
/odom_bridge
/rgb_camera_bridge
/robot_state_publisher
```

- Now we have our `/mars_rover_1` node running.
- We can also retrieve more information about our node with the command `ros2 node info`:

Execute in Terminal #2

- Check that the `heartbeat_executable` is still running.

In [ ]:

```
ros2 node info /mars_rover_1
```

Terminal #2 Output

```
/mars_rover_1
  Subscribers:

  Publishers:
    /parameter_events: rcl_interfaces/msg/ParameterE
vent
    /rosout: rcl_interfaces/msg/Log
  Service Servers:
    /mars_rover_1/describe_parameters: rcl_interface
s/srv/DescribeParameters
    /mars_rover_1/get_parameter_types: rcl_interface
s/srv/GetParameterTypes
    /mars_rover_1/get_parameters: rcl_interfaces/sr
v/GetParameters
    /mars_rover_1/list_parameters: rcl_interfaces/sr
v/ListParameters
    /mars_rover_1/set_parameters: rcl_interfaces/sr
v/SetParameters
    /mars_rover_1/set_parameters_atomically: rcl_int
erfaces/srv/SetParametersAtomically
  Service Clients:

  Action Servers:

  Action Clients:
```

- We will learn what each of these things are throughout this course, don't worry ;).
- But there is still a detail that should bother you....
- In the script, we are still using the `time.sleep(1)` which is **DOES NOT use** the ROS2 clock.
- We have to avoid using system time with ROS2 because time is a vital element for all your programs to work synchronizing.
- To address this we need to introduce **two new concepts**:
  - Custom `Node()` creation.
  - `timer callbacks`.

24/09/2024