ROS 2 Basic Concepts

Estimated time to completion: **12 minutes**

2.9 The ROS 2 Node Class

As you start working with ROS 2 nodes in more complex projects, the recommended approach is to create a **custom ROS 2 node class** that inherits from the **parent `Node` class**.

If you're not yet familiar with classes and inheritance in Python, we recommend referring to the Python 3 Course for a deeper understanding.

In a nutshell:

When our custom class inherits from a **parent class**, it automatically gains all the parent class's functionality, meaning it can do everything the parent class can.

From there, we can **extend** it by adding the specific functionality our application needs.

Find below the part of code that we should add to the script:

In [ ]:

```
from rclpy.node import Node

class HeartbeatNode(Node):
    def __init__(self, rover_name, timer_period=0.2):
        # call super() in the constructor to initialize t
        # the parameter we pass is the node name
        self._rover_name = rover_name
        super().__init__(self._rover_name)
        ...

def main(args=None):
    # initialize the ROS2 communication
    rclpy.init(args=args)
    # declare the node constructor
    node = HeartbeatNode(rover_name="mars_rover_1", timer
    ...
```

- In this case we are creating a class named `HeartbeatNode`, that inherits from `Node`.
- In the init function we have **two arguments that we can modify as needed:
  - `rover_name`: This will be used to name the node.
  - `timer_period`: We will use it for our timer

- Here is the complete code.
- **Please update your code with the following changes**:

heartbeat.py

In [ ]:

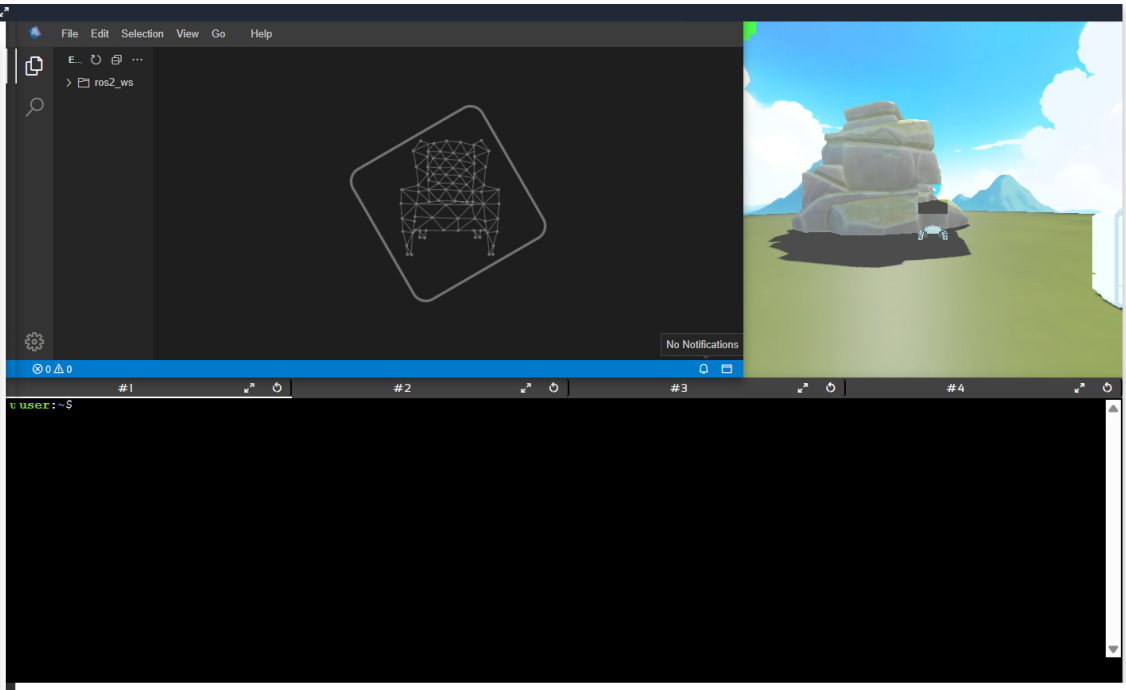
```
#!/usr/bin/env python

import rclpy
from rclpy.node import Node

class HeartbeatNode(Node):
    def __init__(self, rover_name, timer_period=0.2):
        # call super() in the constructor to initialize t
        # the parameter we pass is the node name
        self._rover_name = rover_name
        super().__init__(self._rover_name)
        # create a timer sending two parameters:
        # - the duration between two callbacks (0.2 secon
        # - the timer function (timer_callback)
        self.create_timer(timer_period, self.timer_callba

    def timer_callback(self):
        ros_time_stamp = self.get_clock().now()
        self.get_logger().info(self._rover_name +" is ali

def main(args=None):
    # initialize the ROS2 communication
    rclpy.init(args=args)
    # declare the node constructor
    node = HeartbeatNode(rover_name="mars_rover_1", timer
    # keeps the node alive, waits for a request to kill t
    rclpy.spin(node)
    # shutdown the ROS2 communication
    rclpy.shutdown()

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

Also, create a `timer` object and its `callback`:

In [ ]:

```
self.create_timer(timer_period, self.timer_callback)
...
def timer_callback(self):
        ros_time_stamp = self.get_clock().now()
        self.get_logger().info(self._rover_name +" is ali
```

This `timer` object will trigger the `timer_callback` method every `timer_period` seconds.



In [ ]:

```
rclpy.spin(node)
```

- Another new concept here is `rclpy.spin(node)`.
- The `spin()` function is used to tell ROS2 to process and manage incoming `callbacks` within the given ROS2 node.
- In this case, it primarily manages the `timer_callback`.
- You can see `spin()` as a highly efficient `INFINITE WHILE LOOP`.

- Notes -

`timer` objects are very useful in ROS2. You'll become more familiar with them as you use them throughout the course.

- End of Notes -

Recompile your package once you have made the changes to your code.

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

```
colcon build
```

In [ ]:

```
source ~/ros2_ws/install/setup.bash
```

Now launch the program.

Execute in Terminal #1

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable
```

Terminal #1 Output

```
[INFO] [1724685100.419880909] [mars_rover_1]: mars_rover_1 is alive...Time(nanoseconds=1724685100419063526, clock_type=ROS_TIME)
[INFO] [1724685101.419830285] [mars_rover_1]: mars_rover_1 is alive...Time(nanoseconds=1724685101419063368, clock_type=ROS_TIME)
```

Execute in Terminal #2

In [ ]:

```
source ~/ros2_ws/install/setup.bash
```

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
```

As you can see, the `/mars_rover_1` node now appears in the list and will continue running until you stop it by pressing `CTRL+C` in the terminal where you executed it.

- Exercise 2.2 -

Create a new method and `entry point` so that we can launch at the same time two different nodes:

- In future missions we might need to manage several rovers at the same time.
- In this case, `mars_rover_1` and `mars_rover_2`.
- Create a new method inside `heartbeat.py` similar to `main()`, named `main2()`.
- The only difference should be the **name of the robot**.
- Then add a new `entry point` to allow executing `main2()`.
- Refer to **exercise 2.1** for practice.
- You can now execute both `main` and `main2` simultaneously.

- End of Exercise 2.2 -

- Expected result for Exercise 2.2 -

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

```
colcon build
```

In [ ]:

```
source ~/ros2_ws/install/setup.bash
```

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable
```

Terminal #1 Output

```
[INFO] [1724685096.419794989] [mars_rover_1]: mars_rover_1 is alive...Time(nanoseconds=1724685096419083179, clock_type=ROS_TIME)
```

Execute in Terminal #2

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable2
```

Terminal #2 Output

```
[INFO] [1724685921.079115885] [mars_rover_2]: mars_rover_2 is alive...Time(nanoseconds=1724685921078429065, clock_type=ROS_TIME)
```

Execute in Terminal #3

In [ ]:

```
ros2 node list
```

Terminal #3 Output

```
/clock_bridge
/cmd_vel_bridge
/fake_joint_state_publisher
/laser_bridge
/mars_rover_1
/mars_rover_2
/odom_bridge
/rgb_camera_bridge
/robot_state_publisher
```

- End Expected result for Exercise 2.2 -

- RECOMMENDATION -

PLEASE TRY TO DO THE EXERCISE BEFORE LOOKING THE SOLUTION

- END RECOMMENDATION -

- Solution for Exercise 2.2 -

heartbeat.py

In [ ]:

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

import rclpy
from rclpy.node import Node

class HeartbeatNode(Node):
    def __init__(self, rover_name, timer_period=0.2):
        # call super() in the constructor to initialize t
        # the parameter we pass is the node name
        self._rover_name = rover_name
        super().__init__(self._rover_name)
        # create a timer sending two parameters:
        # - the duration between two callbacks (0.2 secon
        # - the timer function (timer_callback)
        self.create_timer(timer_period, self.timer_callba

    def timer_callback(self):
        ros_time_stamp = self.get_clock().now()
        self.get_logger().info(self._rover_name +" is ali

def main(args=None):
    # initialize the ROS2 communication
    rclpy.init(args=args)
    # declare the node constructor
    node = HeartbeatNode(rover_name="mars_rover_1", timer
    # keeps the node alive, waits for a request to kill t
    rclpy.spin(node)
    # shutdown the ROS2 communication
    rclpy.shutdown()

def main2(args=None):
    # initialize the ROS2 communication
    rclpy.init(args=args)
    # declare the node constructor
    node = HeartbeatNode(rover_name="mars_rover_2", timer
    # keeps the node alive, waits for a request to kill t
    rclpy.spin(node)
    # shutdown the ROS2 communication
    rclpy.shutdown()

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

setup.py

In [ ]:

```python
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
            'heartbeat_executable2 = mars_rover_systems.h
        ],
    },
)
```

- End of Solution for Exercise 2.2 -

- Starting multiple nodes individually is not scalable
- Typical ROS2 systems may require launching dozens of nodes for a single application.
- We need a way to start multiple nodes simultaneously with a single command.
- This is where `launch` files come into play.

24/09/2024