ROS 2 Basic Concepts

Estimated time to completion: **12 minutes**

2.6 Create your first ROS 2 program

Now that you've been introduced to some core ROS 2 concepts, it's time to create your **first ROS 2 program!** For this, you'll be working on a new mission for the Mars Rover robot.

🚀 MISSION: Create a Heartbeat Program for the Mars Rover

One of the first programs in any control mission is a **heartbeat** program. This is a simple system check to ensure that the Rover is transmitting and operational.

In this mission, you'll create a basic program called `heartbeat.py`, which will periodically print a status message.

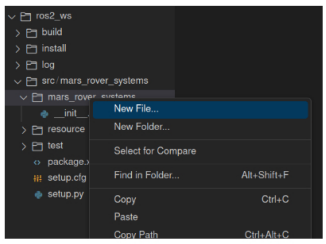
We will place `heartbeat.py` inside the `mars_rover_systems` package so that it can be executed as part of the Mars Rover's system checks.

Let's get started!

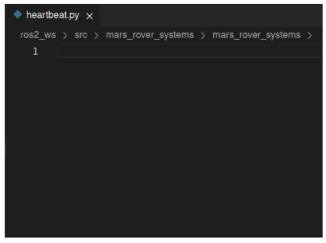
- Example 2.6 -

1. Create a Python file

- Create a Python file in the `mars_rover_systems` directory, which is inside the `mars_rover_systems` package (it may be confusing, but bear with me for now).
- For this exercise, we will create simple Python file named heartbeat.py.
- You can create it directly by **right-clicking** on the IDE in the `mars_rover_systems` directory of your package and selecting `New File`.



A new Tab should have opened in the IDE with empty content.



- Because this is your **FIRST ROS2 program** we will go through it step by step together.
- Please **TYPE** each line of code yourself, as it will help you pay closer attention to each element.
- Please go to the IDE and type the following into the file `heartbeat.py`:

In [ ]:

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

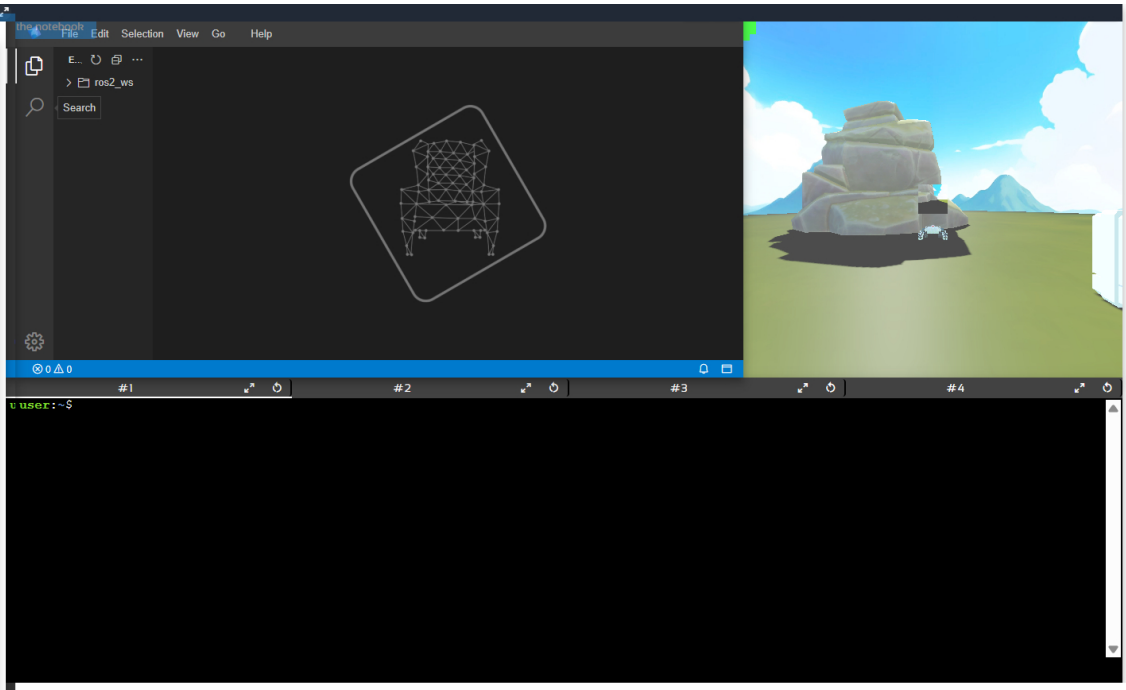
- It's recommended to place a `shebang` (or `hashbang`) on the first line of your Python file
- It is commonly used in script files on Unix-like operating systems (such as Linux and macOS) to indicate **which interpreter should be used to execute the script.**
- ROS2 might encounter issues if you don't specify the interpreter to use in python scripts.

In [ ]:

```
import rclpy
```

Client Libraries

- What is `rclpy`?
- In a nutshell ROS client libraries allow ROS2 scripts written in various programming languages to use ROS2.
- A core ROS client library (RCL) provides the standard functionality needed by various ROS APIs.
- This approach makes it easier to develop language-specific client libraries.



The ROS2 team currently maintains the following client libraries:

- `rclcpp` : ROS2 client library for **C++**.
- `rclpy` : ROS2 client library for **Python**.
- As a point of interest, there is a RUST `ros2_client` that is gaining traction and could become very important in the near future.

Additionally, other client libraries have been developed by the ROS community. You can find more details in this article.

In [ ]:

```
def main(args=None):
```

- We define a method named `main` , which is standard in Python.

In [ ]:

```
rclpy.init(args=args)
```

- We use the **Python client library for ROS2** to initialize the **ROS2 communication and systems** needed for our program.

In [ ]:

```
print("Mars rover 1 is alive...")
```

- We use Python's simple print function to display messages.

In [ ]:

```
rclpy.shutdown()
```

- We close all the ROS2-related system connections that we initiated with `rclpy.init(args=args)` .

In [ ]:

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

- We add the execution of the `main()` method that we defined earlier in the code.
- This will invoke the `main()` method when we run the Python script.

You should have something like this:

heartbeat.py

In [ ]:

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

def main(args=None):
    # initialize the ROS communication
    rclpy.init(args=args)
    # print a message to the terminal
    print("Mars rover 1 is alive...")
    # shutdown the ROS communication
    rclpy.shutdown()

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

- Great! Let's execute it in the simplest way possible:

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws/src/mars_rover_systems/mars_rover_systems/
python3 heartbeat.py
```

Terminal #1 Output

```
Mars rover 1 is alive...
```

- **CONGRATULATIONS!**, You've just executed your first ROS2 program!

- Let's improve this slightly.
- You see, knowing the exact path of each script inside each of the ROS2 packages you might have is an impossible task.
- Robot systems like our Mars Rover can have around thirty different packages, each one with its own set of scripts.
- There must be a better way... and there is!
- The `ros2 run` command that we used earlier for teleoperating our Mars Rover.
- To use it for our own package, we need to make the following updates:

2. Modify the setup.py File to Execute Your Python File.

- The `setup.py` in Python modules, not specific to ROS2, defines the package, handles dependencies, and manages distribution.
- This is common in many Python-based projects for this purpose.
- In ROS2 Python packages, in addition to the above, `setup.py` integrates with the ROS2 ecosystem, ensuring compatibility with ROS2's build tools, dependency management, and node execution.

- We could spend a lot of time explaining the various features of `setup.py` , but we'll focus on the most commonly used features for ROS2 as needed.

- Take a look at the `setup.py` file , specifically at `console_scripts` .

- It is placed inside a dictionary named `entry_points`.

Custom `Entry Points`:

- `setup.py` defines what is known as `entry points`.
- These `entry points` allow ROS2 to recognize and execute Python-based scripts.
- `Entry points` have **two parts**:
  - `entry_point_name`: This is an arbitrary name you provide, which will be used in ROS2.
  - `entry_point_script_path`: This specifies the path to the **METHOD** that you want to execute when calling the **entry point name**.

In [ ]:

```
'console_scripts': [
            'entry_point_name = entry_point_script_path',
            'another_entry_point_name = another_entry_poi
            ...
            'still_another_entry_point_name = still_anoth
        ],
```

- As you can see, **you can include as many as you need**.

- Let's take a look at what we need to include in order to execute the `main()` method in our `heartbeat.py` script.

In [ ]:

```
'console_scripts': [
            'heartbeat_executable = mars_rover_systems.he
        ],
```

- `entry_point_script_path`: `mars_rover_systems.heartbeat:main`:
  - 1) This tells ROS2 that we want to go to a folder inside the package named `mars_rover_systems`. By default when we create a package, a folder with the **EXACT SAME NAME** as the package is generated inside it.
  - 2) It then specifies which script inside the folder `mars_rover_systems` we have to look for, in this case `heartbeat.py`.
  - 3) Within `heartbeat.py` (note the use of a `:` instead of a `.`) we specify the method to be executed, in this case, the method named `main()`.

Your final `setup.py` file should look like this:

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
        ],
    },
)
```

- This will allow us to execute the script using the `ros2 run` command, without needing to know the exact path where the script is located:

In [ ]:

```
ros2 run <name_of_the_package> <entry_point_name>
```

- In our case:

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable
```

- Let's **COMPILE** to apply the latest changes:

Execute in Terminal #1

- RECOMMENDATION -

PLEASE TYPE THE COMMANDS, as this will help you better memorize and understand what you are doing.

- END RECOMMENDATION -

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

```
colcon build
```

In [ ]:

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

- And now run it:

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

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

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable
```

If everything goes as planned, you should see an output similar to the following:

Terminal #1 Output

```
Mars rover 1 is alive...
```

- Exercise 2.1 -

Create a New Method and Modify `setup.py` to Execute it:

- Add a new method to `heartbeat.py` similar to `main()`, named `main_shutdown()`.
- This method should print a message saying `Shutting down Mars rover 1...`.

- End of Exercise 2.1 -

- Expected result for Exercise 2.1 -

Terminal #1 Output

```
Shutting down Mars rover 1...
```

- TIP -

- If you press the **TAB** key TWICE in succession, the terminal will autocomplete or show you available options.
- So if you type this and then press TAB twice:

Execute in Terminal #1

In [ ]:

```
ros2 run mars_rover_systems
```

Terminal #1 Output

```
user:~/ros2_ws$ ros2 run mars_rover_systems
--prefix                    heartbeat_executable
heartbeat_executable_shutdown
```

- It will show the **TWO available** `entry point` **names**.
- If it doesn't, something may have gone wrong when you created the new `entry point` in setup.py, or the package might not have been compiled correctly. This is a good way to verify that ROS2 is working correctly.

- END TIP -

- End Expected result for Exercise 2.1 -

- RECOMMENDATION -

PLEASE TRY TO DO THE EXERCISE BEFORE LOOKING AT THE SOLUTION

- END RECOMMENDATION -

- Solution for Exercise 2.1 -

heartbeat.py

In [ ]:

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

```python
import rclpy

def main(args=None):
    # initialize the ROS communication
    rclpy.init(args=args)
    # print a message to the terminal
    print("Mars rover 1 is alive...")
    # shutdown the ROS communication
    rclpy.shutdown()

def main_shutdown(args=None):
    rclpy.init(args=args)
    print("Shutting down Mars rover 1...")
    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
            'heartbeat_executable_shutdown = mars_rover_s
        ],
    },
)
```

- Compile:

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

```
colcon build
```

In [ ]:

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

- And now run it:

Execute in Terminal #1

In [ ]:

```
cd ~/ros2_ws
```

In [ ]:

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

In [ ]:

```
ros2 run mars_rover_systems heartbeat_executable_shutdown
```

- End of Solution for Exercise 2.1 -

24/09/2024