First Action

Polytech Angers - Mobile Robotics

In the later you will need a new action interface. Let's create it now.

> Note that this is done with a pedagogic objective. You should use standard interface as much as possible and avoid creating your own when possible !

Create the action message

To create an action in your nodes, you may need to have the action interface

Note that as the moment it is not possible to create an action interface within a python package. Fortunately, the tp1 package is not a python package... How lucky...

In the tp1 package, create a new directory named `action` (next to the `src` and `launch` directory).

```
docker@ros2:~/wdir/src/tp1$ mkdir action
```

In this `action` directory, create a file named `Path.action` with the following content:

```
geometry_msgs/Point[] path
---
geometry_msgs/Point final_point
---
geometry_msgs/Point reached_point
```

Here is the `tp1` directory you should have:

```
.
├── action
│   └── Path.action
├── CMakeLists.txt
├── launch
│   └── my_launch.py
├── package.xml
└── src
    └── my_teleop.py
```

To reminder, the first element is the goal type, the second is the result and the last one is the feedback. **Note that the name of the file has to start with a capital letter!**

To be able to build this package and use this action interface, it is needed to add the following lines into the `CMakeLists.txt` file (to pass the definition to the rosidl code generation pipeline) just after the two `find_package()`:

```
find_package(rosidl_default_generators REQUIRED)
find_package(geometry_msgs REQUIRED)
    rosidl_generate_interfaces(${PROJECT_NAME}
    "action/Path.action"
    DEPENDENCIES geometry_msgs
)
```

and update the `install()` command by adding the `action` directory:

```
install(DIRECTORY launch action DESTINATION share/${PROJECT_NAME})
```

It is also needed to indicate that this package depends on the `action_msgs` package and the `geometry_msgs` package. To do that, you need to add the following in the `package.xml` file before the `<export>` tag:

```
<buildtool_depend>rosidl_default_generators</buildtool_depend>
<depend>action_msgs</depend>
<depend>geometry_msgs</depend>
<member_of_group>rosidl_interface_packages</member_of_group>
```

Try to build the package to check if everything is OK so far...

Create the action server

In your package `tp1`, create a new node named `test_path.py` (do not forget to make it executable). Fill the `test_path.py` file with the following content:

```
#!/bin/python3

import time # needed for the sleep

# ROS2 lirabries
import rclpy
from rclpy.action import ActionServer  # needed for the action server
from rclpy.node import Node

# The considered interface for the action
```

```
from tp1.action import Path

class PathActionServer(Node):
    """
    Class to define a new node, that will provide the action server
    """

    action_server: rclpy.action.server.ActionServer  # The action server
for this node

    def __init__(self):
        super().__init__('path_action_server')
        self.action_server = ActionServer(
            self,
            Path,               # The type of the action
            'path',             # The action name
            self.follow_path)   # The callback function

    def follow_path(self,
goal_handle:rclpy.action.server.ServerGoalHandle):
        """
        The callback function, called when receiving a goal for the /path
action
        goal_handle: the goal
        """
        self.get_logger().info('Executing goal...')

        # variable to handle the feedback (the messages sent before the end
of the action)
        feedback_msg = Path.Feedback()

        # for this example we just loop over all the point of the path
        for point in goal_handle.request.path:
            # for each point, we define the feedback point as the current
point
            feedback_msg.reached_point = point
            # we publish the feedback
            goal_handle.publish_feedback(feedback_msg)
            # we wait for 1 second
            time.sleep(1)

        # to indicate that the goal was successful (otherwise it is assumed
aborted by default)
        goal_handle.succeed()

        # variable to handle the resulting message
        result = Path.Result()
        # for this example it is just the last point of the path
        result.final_point = goal_handle.request.path[-1]
        self.get_logger().info('...Finished')
        return result


def main(args=None):
```

```
    rclpy.init(args=args)
    path_action_server = PathActionServer()
    try:
        rclpy.spin(path_action_server)
    except KeyboardInterrupt:
        pass


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

To build the package `tp1` you can do:

```
colcon build --symlink-install --packages-select tp1
```

The `--package-select` option allows to define the package you want to build, instead of building all the packages of your workspace.

Once this is done you should be able to start the node:

```
docker@ros2:~/wdir$ ros2 run tp1 test_path.py
```

To test if everything is OK you can publish a goal with the `send_goal` tool:

```
ros2 action send_goal /path tp1/action/Path "{path:[{x: 0.0,y: 0.0,z: 0.0},
{x: 1.0,y: 0.0,z: 0.0},{x: 0.0,y: 1.0,z: 0.0},{x: 0.0,y: 0.0,z: 1.0}]}" --
feedback
```

Here are the explanation of the parameters:

- `/path` the name of the action (`ros2 action list` command to display all the available actions)
- `tp1/action/Path` The type of the action (interface: `ros2 action list -t` to display the type of the actions)
- `"{path:[{x: 0.0,y: 0.0,z: 0.0},{x: 1.0,y: 0.0,z: 0.0},{x: 0.0,y: 1.0,z: 0.0},{x: 0.0,y: 0.0,z: 1.0}]}"` The value of the action goal. This should be consistent with the type:

```
docker@ros2:~/wdir$ ros2 interface show tp1/action/Path
geometry_msgs/Point[] path
        float64 x
        float64 y
        float64 z
---
geometry_msgs/Point final_point
```

```
        float64 x
        float64 y
        float64 z
---
geometry_msgs/Point reached_point
        float64 x
        float64 y
        float64 z
```

Here we have a list of Point, and each point has an x, y and z value.

- The `--feedback` option is to display the feedback from the node.

You should have the following result:

```
docker@ros2:~/wdir$ ros2 action send_goal /path tp1/action/Path "{path:[{x:
0.0,y: 0.0,z: 0.0},{x: 1.0,y: 0.0,z: 0.0},{x: 0.0,y: 1.0,z: 0.0},{x: 0.0,y:
0.0,z: 1.0}]}" --feedback
Waiting for an action server to become available...
Sending goal:
     path:
- x: 0.0
  y: 0.0
  z: 0.0
- x: 1.0
  y: 0.0
  z: 0.0
- x: 0.0
  y: 1.0
  z: 0.0
- x: 0.0
  y: 0.0
  z: 1.0

Goal accepted with ID: ad9812f1c7954a478f07389b9edfec17

Feedback:
    reached_point:
  x: 0.0
  y: 0.0
  z: 0.0

Feedback:
    reached_point:
  x: 1.0
  y: 0.0
  z: 0.0

Feedback:
    reached_point:
  x: 0.0
  y: 1.0
```

```
  z: 0.0

Feedback:
    reached_point:
  x: 0.0
  y: 0.0
  z: 1.0

Result:
    final_point:
  x: 0.0
  y: 0.0
  z: 1.0

Goal finished with status: SUCCEEDED
```

Now everything is ready for the next node : follow path.