ROS Programming – Advanced Topics

Stefano Carpin

Department of Computer Science and Engineering
School of Engineering
University of California, Merced
`https://sites.ucmerced.edu/scarpin`
`https//robotics.ucmerced.edu`

Remapping: changing node names

- process to change some standard arguments in ROS applications
- example: node name. Why would you change it?
  - having multiple instances of the same node with the same name is allowed, but will most likely create problems
- `ros2 run talklisten talker --ros-args --remap __name:=newtalker`
- the output will be

```
[INFO] [1673935144.419936102] [newtalker]: Sending message #1
[INFO] [1673935145.409056027] [newtalker]: Sending message #2
[INFO] [1673935146.409073411] [newtalker]: Sending message #3
[INFO] [1673935147.409061516] [newtalker]: Sending message #4
[INFO] [1673935148.408931236] [newtalker]: Sending message #5
```

Remapping: changing topic names

- often times the functionality of a node is the same, but just the topic name changes

```
ros2 run packagename executablename --ros-args --remap
   originaltopicname:=newtopicname
```

- we already saw this in chapter 4

Namespaces

- introduced to avoid name clashes (like in C++ and other languages)
- example: a robot with two cameras runs two instances of the same node to process the two separate input streams. How can we separate the output streams, too?
- namespaces allow to solve this program without changing the source
- namespaces allow to change the names of resources (nodes names, topics, etc) by prepending a unique suffix that can be specified when the executable is started

```
ros2 run talklisten talker --ros-args --remap __ns:=/t1
```

- now `talker` publishes to a topic called `/t1/message` (even though in the source we call it `message`)

Namespaces (cont'd)

- if we now run the listener, it will not print anything b/c it subscribes to `message`
- this can be fixed by remapping the listener, too

```
ros2 run talklisten listener --ros-args --remap
message:=/t1/message
```

- remappings can be specified in launch files, too (example: `t1remap.launch.xml`)

ROS names

- having two resources with exactly the same name is legit, but will likely create problems
- Example: we can run two instance of `talker` from the `demo_nodes_cpp` package
- `ros2 node list` tells us this is a bad idea

```
WARNING: Be aware that are nodes in the graph that share an
exact name,  this can have unintended side effects.
/talker
/talker
```

- How can we fix it?
- Remapping the name with `-remap __name` is a partial fix because they continue to publish to the same topic (may or may not be what we want)

ROS names

- the solution is to remap the namespaces for one of the nodes
- `ros2 run demo_nodes_cpp talker -ros-args -remap __ns:=/t2`
- `ros2 node list` gives

```
/t2/talker
/talker
```

- `ros2 node list` give

```
/chatter
/t2/chatter
```

- Success! Now they are fully separated

ROS fully qualified names

- ROS resources (topics, service, etc.) have *relative* and *fully qualified names*
- a fully qualified name starts with / and identifies the namespace within which the resource exist
- relative names are referred to the namespace of the nodes that create them.
- relative names *do not* start with /
- if a resource is create without setting a namespace, its fully qualified name is the name of the resource preceded by the / character
- if a resource is created within a namspace, its fully qualified name is the name of the resource preceded by the name of the namespace
- Exercise: go back and revisit the listings in chapter 3

Parameters

- names and namespaces are associated with all nodes
- ROS parameters allow the user to define new application-specific parameters
- Ex: on which port is a sensor connected? With which frequency should a sensor be queried?
- We may wish to set them when a node starts, or even change them during execution
- `rclcpp` provides methods to set, retrieve, and change parameters from within an executable
- parameters can also be modified from the command line
- `ros2 param list` shows all parameters associated with running nodes

Parameters

- start both the `talker` and `listener` nodes from the `demo_nodes_cpp` and run `ros2 param list`
- output

```
/listener:
qos_overrides./parameter_events.publisher.depth
qos_overrides./parameter_events.publisher.durability
qos_overrides./parameter_events.publisher.history
qos_overrides./parameter_events.publisher.reliability
start_type_description_service
use_sim_time
/talker:
qos_overrides./parameter_events.publisher.depth
qos_overrides./parameter_events.publisher.durability
qos_overrides./parameter_events.publisher.history
qos_overrides./parameter_events.publisher.reliability
start_type_description_service
use_sim_time
```

- all nodes have a parameter called `use_sim_time` (more later)

Interacting with parameters from the command line

- use `ros2 param describe` to get information about an existing parameter
- `ros2 param describe talker use_sim_time`
- output:

```
Parameter name: use_sim_time
Type: boolean
Constraints:
```

- use `ros2 param get` to retrieve the value of an existing parameter
- `ros2 param get talker use_sim_time`
- output:

```
Boolean value is: False
```

- parameters are local and not global (each node has its own copy of `use_sim_time`

Interacting with parameters from the command line

- the value of a parameter can also be set from the command line
- use `ros2 param set` to retrieve the value of an existing parameter
- `ros2 param set talker use_sim_time True`
- verify that the parameter has changed!
- general syntax

  `ros2 param set node_name parameter_name value`

Interacting with parameters via `rclcpp`

- In your code, you first declare a parameter and then you can retrieve its value
- parameters have a name (string) and a type that must be specified when they are declared; default values must be given
- Code example: `paramclient.cpp`
- parameters can be changed from the outside in various ways, e.g. when an executable starts
- `ros2 run examples paramclient -ros-args -p "sensorport":="/dev/USB0"`
- parameters can also be changed from the command line
- `ros2 param set paramclient sensorport /dev/USB1`
- if you use `ros2 param set`, parameters must be set after the node is running
- try it and you'll see the value changes

More about parameters

- Parameters can also be specified in launch files
- Example: `param.launch.xml`
- nodes can also change parameters using `set_parameter_value` (mirrors `get_parameter_value`)

Loading and saving parameters

- All parameters can be saved to a YAML file that can then be edited, if needed
  `ros2 param dump <nodename>`
- Parameters can also be loaded at run time after a node starts
  `ros2 param load <nodename> <parameterfile>`
- we can load the parameters from a YAML file when the node starts, e.g.,
  `ros2 run talklisten listener -ros-args -params-file parameters.yaml`
- the same can be specified in the launch file, see e.g., `paramfile.launch.xml`

Intercepting parameters changes on-the-fly

- First example performs busy waiting to get the parameter – not practical
- to be notified when a parameter changes, subscribe to the topic `parameter_events` of type `rcl_interfaces/msg/ParameterEvent`

rcl_interfaces/msg/ParameterEvent

```
# The time stamp when this parameter event occurred.
builtin_interfaces/Time stamp

# Fully qualified ROS path to node.
string node

# New parameters that have been set for this node.
Parameter[] new_parameters

# Parameters that have been changed during this event.
Parameter[] changed_parameters

# Parameters that have been deleted during this event.
Parameter[] deleted_parameters
```

Intercepting parameters changes on-the-fly

- Example: `paramevent.cpp`

The parameter `use_sim_time`

- all nodes have a parameter called `use_sim_time`
- useful when the node interacts with a simulator (e.g., Gazebo)
- if `use_sim_time` is `True`, the node retrieves time from the simulator and not from the system clock
- useful when a node must reason about time (e.g., when using `tf2`)

Calling Services

- we already saw the commands `ros2 service list` and how `ros2 interface` can be used to learn about a service
- For example `ros2 interface show sensor_msgs/srv/SetCameraInfo` shows

```
sensor_msgs/CameraInfo camera_info # The camera_info to store
---
bool success                      # True if the call succeeded
string status_message         # Used to give details about success
```

- example: `servicecall.cpp`

OOP in ROS

- all code written so far used instances of `rclcpp::Node`
- legit but does not scale to larger applications
- problem: logic governing nodes is not associated with the node itself (we had to use global variables)
- solution: inherit from the class `rclcpp::Node` and use encapsulation
- Example: `talkeroop.cpp`
- later on we'll see how to trigger producers from timers
- Example: `listeneroop.cpp`