EPOS4 / ROS 2
ros1_bridge

Application notes on how to use maxon EPOS4 Positioning Controllers with ROS 2 through ros1_bridge and ROS 1

Author:
Cyril Jourdan
ROS consulting & engineering partner of www.maxongroup.com

Important notes:

This document and all provided sample code has been developed on behalf of maxon motor ag. Any installation steps and code samples are intended for testing purposes only.

Please adapt the code to the needs of your concrete application.

Any warranty for proper functionality is excluded and has to be ensured based on tests within the concrete system environment.

Take care of the wiring and safety instructions mentioned by the "Hardware Reference" of the corresponding controller!

Please submit a request on maxon's Support Center (-> http://support.maxongroup.com) in case of any questions concerning the controller's setup, wiring, or testing.

Edition 2022-03

Table of contents

1. About this Document

1.1. Intended Purpose

The purpose of this document is to help you integrate maxon EPOS4 controllers with ROS 2 (Robotic Operating System 2: https://docs.ros.org/en/foxy/index.html) using the software package `ros1_bridge`. The present documentation goes along with the ROS package `maxon_epos4_ros2` as an example to get started.

This documentation focuses on bridging `ros_canopen` (http://wiki.ros.org/ros_canopen) topics and services but it could also be used as a guideline to bridge other ways of controlling the EPOS4 with ROS 1.

1.2. Target Audience

This document is written for both beginners and advanced users who have basic knowledge of CANopen (https://www.can-cia.org/canopen/), ROS 1 and ROS 2. It contains all the necessary links to extend your knowledge on the topics, so it mainly focuses on specificities related to `ros1_bridge` setup and ROS 2.

1.3. Use cases

ROS2 has been developed while considering the real time requirements of multi-axis robotic systems. However, `ros_canopen` has not been officially migrated to ROS 2 yet, so an approach using `ros1_bridge` will be used. This approach is suitable for non-real-time applications where it is acceptable to control the axis independently.

The code examples contained in the `maxon_epos4_ros2` package describe simple use cases using one controller, which can be easily extended to more controllers for a specific application.

1.4. Conventions

All the commands that need to be executed in a terminal are written in bold monospace font and start with a "**$**" symbol which should not be typed. Some commands might take two lines on the documentation but should still be entered as one line. Here is an example of a command:

```
$ git clone  https://github.com/Roboprotos/maxon_epos4_ros2
```

File names and ROS package names are written in monospace font, for example: `package.xml`, `maxon_epos4_ros2`.

Two signs are used to attract attention on specific topics:

> ⚠ The warning sign is used for important information.

> 💡 The bulb sign is used for tips.

The following abbreviations are used in the documentation:

- ROS: Robot Operating System
- CAN: Controller Area Network
- PPM: Profile Position Mode

2. EPOS4 and ROS 1 setup

Just follow the documentation from the package `maxon_epos4_ros1` to setup the EPOS4 and ROS 1 with `ros_canopen`: https://github.com/Roboprotos/maxon_epos4_ros1. Current tests have also been done with ROS 1 Melodic.

3. ROS 2 setup

The current documentation has been validated on the NVIDIA Jetson TX2 running Ubuntu 18.04.

To get `ros1_bridge` working properly you can either build ROS 2 Foxy from source or use prebuilt binaries. We will present the first method as it allows more flexibility for projects who also need to bridge custom topics or services. You will need to build ROS 2 without `ros1_bridge` in the first place, so please read this entire paragraph before starting to build.

You can get the ROS 2 Foxy build documentation on the following link: https://docs.ros.org/en/foxy/Installation/Ubuntu-Development-Setup.html. When you are at the step called "Build the code in the workspace", replace the build command with this one to avoid building `ros1_bridge`:

```
$ colcon build --symlink-install --packages-skip ros1_bridge
```

More information can be found here: https://github.com/ros2/ros1_bridge

You then need to uninstall manually the `controller_manager_msgs` package from ROS melodic, starting first with the `share` directory:

```
$ cd /opt/melodic/share
$ sudo rm -r controller_manager_msgs
```

Repeat those steps with `include` directory:

```
$ cd /opt/melodic/include
$ sudo rm -r controller_manager_msgs
```

After that you can build `ros1_bridge` following the official tutorial (https://github.com/ros2/ros1_bridge):

```
$ colcon build --symlink-install --packages-select ros1_bridge --
cmake-force-configure
```

Don't forget to reinstall the `controller_manager_msgs` package:

```
$ sudo apt install ros-melodic-controller-manager-msgs
```

Finally, you need to create a workspace. You can find the instructions on the following link: https://docs.ros.org/en/foxy/Tutorials/Workspace/Creating-A-Workspace.html

4. maxon EPOS4 ROS 2 package

4.1. Overview

The `maxon_epos4_ros2` package contains this documentation and one ROS 2 package:

- `maxon_epos4_ros1_bridge`: this package contains a C++ and a Python examples to show how to use ROS 2 topics and services to control the EPOS4 which is connected to ROS 1 nodes using `ros_canopen`.

> In order to use this documentation with your own project, it is recommended to get one controller working in the desired mode and then either modify the package to fit your needs or write a new one from scratch.

Here is the tree of the package:

```
maxon_epos4_ros2
├── documentation
│   └── maxon_EPOS4_ROS2_ros1_bridge_Documentation.pdf
├── maxon_epos4_ros1_bridge
│   ├── CMakeLists.txt
│   ├── LICENSE
│   ├── maxon_epos4_ros1_bridge
│   │   └── __init__.py
│   ├── package.xml
│   ├── scripts
│   │   └── python_example_1dof_ppm.py
│   └── src
│       └── cpp_example_1dof_ppm.cpp
└── README.md
```

Some files specify `1dof` and `ppm` which stands for 1 degree of freedom in Profile Position Mode. The `CMakelists.txt` file defines how to build and install the package (https://docs.ros.org/en/foxy/How-To-Guides/Ament-CMake-Documentation.html), while its properties are written in `package.xml`. The `maxon_epos4_ros1_bridge` folder containing the empty `__init__.py` file is there only so that the package can be treated as a Python package (see https://docs.ros.org/en/foxy/How-To-Guides/Ament-CMake-Python-Documentation.html).

4.2. Package Installation

Place a copy of the `maxon_epos4_ros2` metapackage under your ROS 2 workspace source folder, for example located here:

```
~/dev_ws/src/maxon_epos4_ros2
```

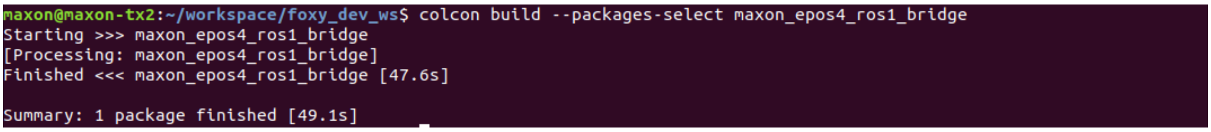
To do that simply go to the workspace source directory and clone the package from GitHub using git:

```
$ cd ~/dev_ws/src
$ git clone https://github.com/Roboprotos/maxon_epos4_ros2
```

Go back to the workspace folder to build the `maxon_epos4_ros1_bridge` package contained in the metapackage (this step is needed for building the C++ example code and installing the Python node):

```
$ source ~/dev_ws/install/setup.bash
$ cd ~/dev_ws
$ colcon build --packages-select maxon_epos4_ros1_bridge
```

You should get the following output:



```
maxon@maxon-tx2:~/workspace/foxy_dev_ws$ colcon build --packages-select maxon_epos4_ros1_bridge
Starting >>> maxon_epos4_ros1_bridge
[Processing: maxon_epos4_ros1_bridge]
Finished <<< maxon_epos4_ros1_bridge [47.6s]

Summary: 1 package finished [49.1s]
```

You can check that the two executables are present in the following `lib` folder:

```
$ cd ~/dev_ws/install/maxon_epos4_ros1_bridge/lib
```

There you should find a `maxon_epos4_ros1_bridge` folder with the two executables: `cpp_example_1dof_ppm` and `python_example_1dof_ppm.py`.

Don't forget to rebuild the package after changing the C++ **or Python** node so that the "`ros2 run`" command can find and use the new node.

4.3. maxon_epos4_ros1_bridge package

4.3.1. Building files

The package contains nodes written in both C++ and Python. As a result, its building structure is different from a pure Python ROS 2 package which contains a `setup.py` and `setup.cfg` files but is based on a C++ package with some specific configuration for Python. Therefore, everything is handled in the `CMakeLists.txt` and `package.xml` files.

It is necessary to have the following lines in the `package.xml` file to deal with Python nodes:

```
<buildtool_depend>ament_cmake_python</buildtool_depend>
<exec_depend>rclpy</exec_depend>
```

Regarding the `CMakeLists.txt` file, the following lines are necessary to get the Python nodes installed properly when calling “`colcon build`” command, otherwise ROS 2 can’t find the node when using the “`ros2 run`” command:

```
# Find dependencies
find_package(ament_cmake_python REQUIRED)
find_package(rclpy REQUIRED)
...
...
# Python nodes
ament_python_install_package(${PROJECT_NAME})

# Install Python executables
install(PROGRAMS
  scripts/python_example_1dof_ppm.py
  DESTINATION lib/${PROJECT_NAME}
)
```

4.3.2. Code examples

The source folder `src` contains a code example written in C++, and the folder `scripts` contains one written in Python.

4.3.2.1. C++ example

The `cpp_example_1dof_ppm.cpp` file shows you how to call the various driver services (init, halt, recover and shutdown) of a controller. It also shows how to publish Target Position value to the command topic and how to subscribe to the Position Actual Value stored in the `JointStates` topic. It contains some comments to get a better understanding of the various steps.

4.3.2.2. Python example

The `python_example_1dof_ppm.py` file, written in Python and located in the `scripts` folder, shows how to send a discrete wave of target positions using a sinusoidal function. Further explanation can be found in the various comments in the code.

5. Running the examples

Make sure that your EPOS4 controllers are powered, and that your motors can move safely before running the examples. Also check that your SocketCAN interface is up and running.

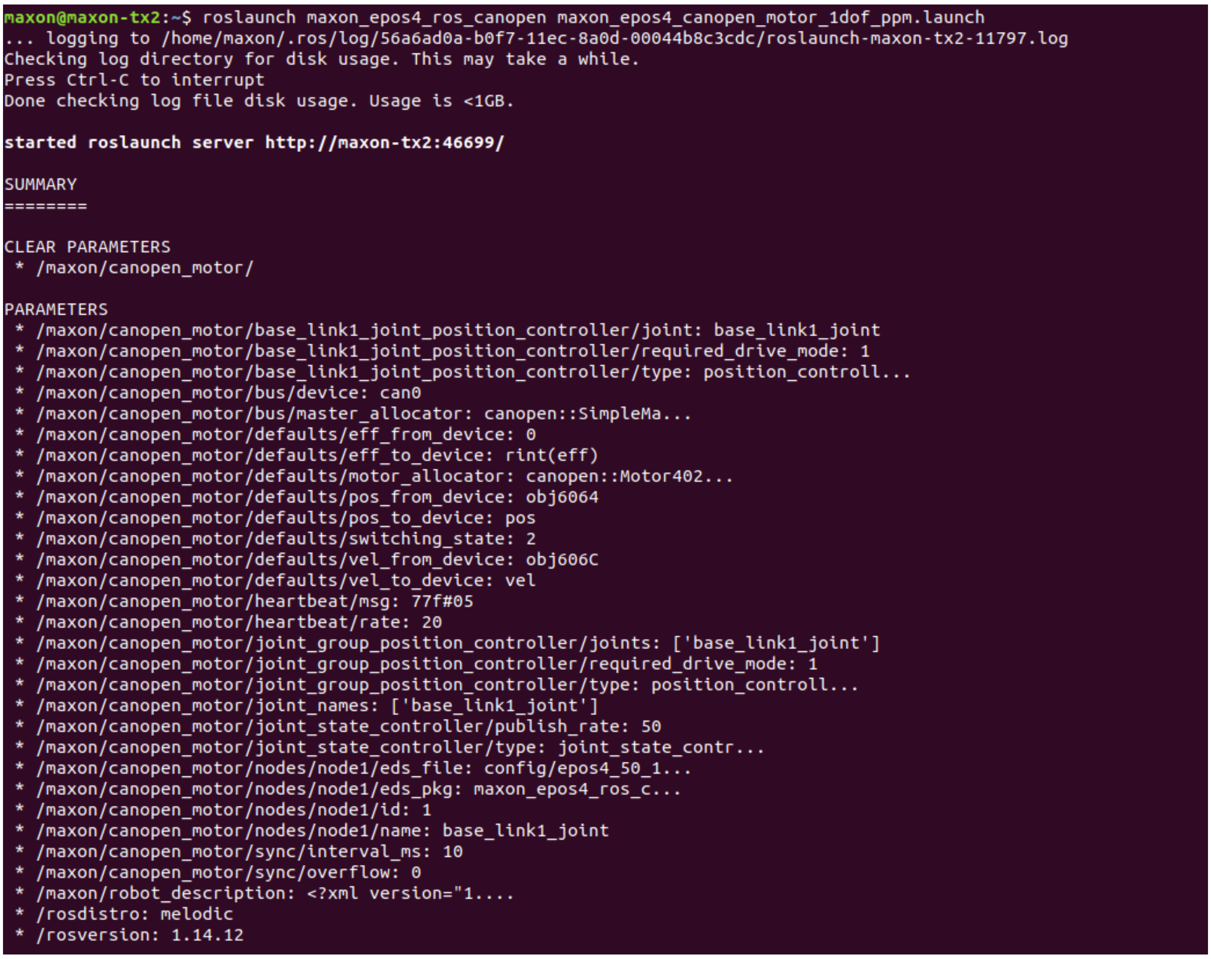
Follow the documentation from the package `maxon_epos4_ros1` and check that you can run the simple example using 1 DOF in PPM. This checking is necessary to make sure that the communication with EPOS4 is working with ROS 1 and `ros_canopen`.

5.1. Using terminal commands

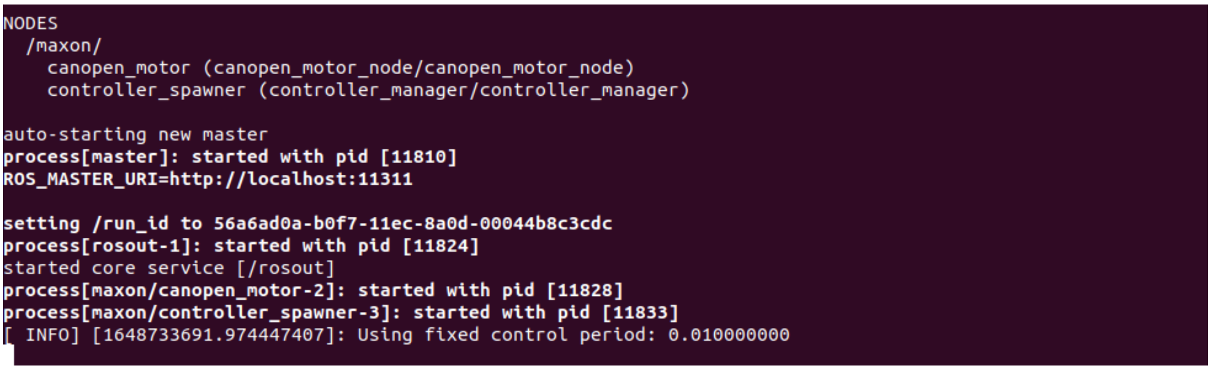
Open a terminal, source ROS 1 and execute the following `roslaunch` command:

```
$ source ~/catkin_ws/devel/setup.bash
$ roslaunch maxon_epos4_ros_canopen
maxon_epos4_canopen_motor_1dof_ppm.launch
```

You should get the following output:



```
maxon@maxon-tx2:~$ roslaunch maxon_epos4_ros_canopen maxon_epos4_canopen_motor_1dof_ppm.launch
... logging to /home/maxon/.ros/log/56a6ad0a-b0f7-11ec-8a0d-00044b8c3cdc/roslaunch-maxon-tx2-11797.log
Checking log directory for disk usage. This may take a while.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://maxon-tx2:46699/

SUMMARY
========

CLEAR PARAMETERS
 * /maxon/canopen_motor/

PARAMETERS
 * /maxon/canopen_motor/base_link1_joint_position_controller/joint: base_link1_joint
 * /maxon/canopen_motor/base_link1_joint_position_controller/required_drive_mode: 1
 * /maxon/canopen_motor/base_link1_joint_position_controller/type: position_controll...
 * /maxon/canopen_motor/bus/device: can0
 * /maxon/canopen_motor/bus/master_allocator: canopen::SimpleMa...
 * /maxon/canopen_motor/defaults/eff_from_device: 0
 * /maxon/canopen_motor/defaults/eff_to_device: rint(eff)
 * /maxon/canopen_motor/defaults/motor_allocator: canopen::Motor402...
 * /maxon/canopen_motor/defaults/pos_from_device: obj6064
 * /maxon/canopen_motor/defaults/pos_to_device: pos
 * /maxon/canopen_motor/defaults/switching_state: 2
 * /maxon/canopen_motor/defaults/vel_from_device: obj606C
 * /maxon/canopen_motor/defaults/vel_to_device: vel
 * /maxon/canopen_motor/heartbeat/msg: 77f#05
 * /maxon/canopen_motor/heartbeat/rate: 20
 * /maxon/canopen_motor/joint_group_position_controller/joints: ['base_link1_joint']
 * /maxon/canopen_motor/joint_group_position_controller/required_drive_mode: 1
 * /maxon/canopen_motor/joint_group_position_controller/type: position_controll...
 * /maxon/canopen_motor/joint_names: ['base_link1_joint']
 * /maxon/canopen_motor/joint_state_controller/publish_rate: 50
 * /maxon/canopen_motor/joint_state_controller/type: joint_state_contr...
 * /maxon/canopen_motor/nodes/node1/eds_file: config/epos4_50_1...
 * /maxon/canopen_motor/nodes/node1/eds_pkg: maxon_epos4_ros_c...
 * /maxon/canopen_motor/nodes/node1/id: 1
 * /maxon/canopen_motor/nodes/node1/name: base_link1_joint
 * /maxon/canopen_motor/sync/interval_ms: 10
 * /maxon/canopen_motor/sync/overflow: 0
 * /maxon/robot_description: <?xml version="1....
 * /rosdistro: melodic
 * /rosversion: 1.14.12
```



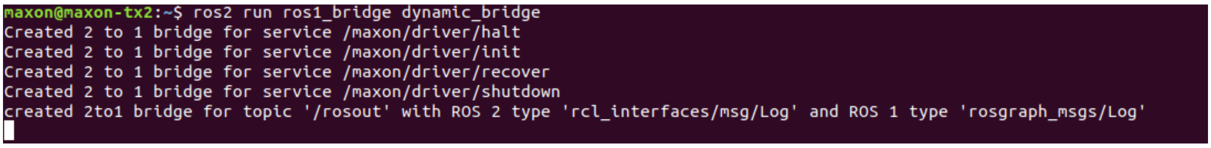
```
NODES
  /maxon/
    canopen_motor (canopen_motor_node/canopen_motor_node)
    controller_spawner (controller_manager/controller_manager)

auto-starting new master
process[master]: started with pid [11810]
ROS_MASTER_URI=http://localhost:11311

setting /run_id to 56a6ad0a-b0f7-11ec-8a0d-00044b8c3cdc
process[rosout-1]: started with pid [11824]
started core service [/rosout]
process[maxon/canopen_motor-2]: started with pid [11828]
process[maxon/controller_spawner-3]: started with pid [11833]
[ INFO] [1648733691.974447407]: Using fixed control period: 0.010000000
```

The `roslaunch` command also calls `roscore` if it hasn't been done previously.

Open a second terminal, source ROS 2 and run the `dynamic_bridge` node from `ros1_bridge`:

```
$ source ~/dev_ws/install/setup.bash
$ ros2 run ros1_bridge dynamic_bridge
```

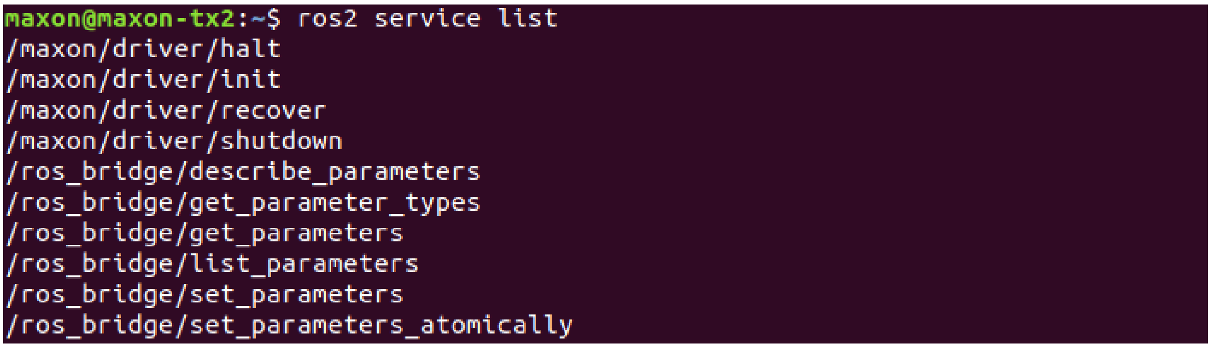
In the output you should see that it has created bridges for the driver services:



```
maxon@maxon-tx2:~$ ros2 run ros1_bridge dynamic_bridge
Created 2 to 1 bridge for service /maxon/driver/halt
Created 2 to 1 bridge for service /maxon/driver/init
Created 2 to 1 bridge for service /maxon/driver/recover
Created 2 to 1 bridge for service /maxon/driver/shutdown
created 2to1 bridge for topic '/rosout' with ROS 2 type 'rcl_interfaces/msg/Log' and ROS 1 type 'rosgraph_msgs/Log'
```

In a third terminal you can first check that ROS 2 can see ROS 1 services:

```
$ source ~/dev_ws/install/setup.bash
$ ros2 service list
```

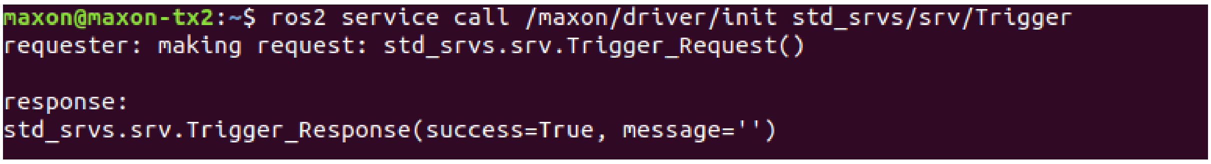


```
maxon@maxon-tx2:~$ ros2 service list
/maxon/driver/halt
/maxon/driver/init
/maxon/driver/recover
/maxon/driver/shutdown
/ros_bridge/describe_parameters
/ros_bridge/get_parameter_types
/ros_bridge/get_parameters
/ros_bridge/list_parameters
/ros_bridge/set_parameters
/ros_bridge/set_parameters_atomically
```

In the same terminal you can then initialize the controller using this command:

```
$ ros2 service call /maxon/driver/init std_srvs/srv/Trigger
```

This should display the following output:



```
maxon@maxon-tx2:~$ ros2 service call /maxon/driver/init std_srvs/srv/Trigger
requester: making request: std_srvs.srv.Trigger_Request()

response:
std_srvs.srv.Trigger_Response(success=True, message='')
```

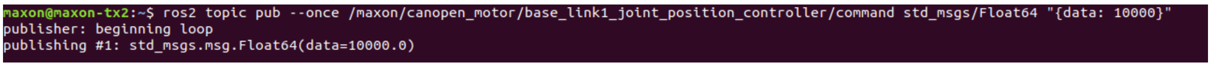
Your first terminal should indicate “Initializing successful” and your EPOS4 LED should go from flashing green to solid green.

Similarly, you can call other services, such as `halt`, `recover` and `shutdown`:

```
$ ros2 service call /maxon/driver/halt std_srvs/srv/Trigger

$ ros2 service call /maxon/driver/recover std_srvs/srv/Trigger

$ ros2 service call /maxon/driver/shutdown std_srvs/srv/Trigger
```

You can then send a command to the motor in a fourth terminal. This is done by publishing data on a controller topic. The following command will send a Target Position of 10000 inc (if “inc” is the chosen position unit on the ROS 1 side) to the first node:

```
$ ros2 topic pub --once
/maxon/canopen_motor/base_link1_joint_position_controller/command
std_msgs/Float64 "{data: 10000}"
```



```
maxon@maxon-tx2:~$ ros2 topic pub --once /maxon/canopen_motor/base_link1_joint_position_controller/command std_msgs/Float64 "{data: 10000}"
publisher: beginning loop
publishing #1: std_msgs.msg.Float64(data=10000.0)
```

In the `ros1_bridge` terminal you can see that a bridge is created for the command topic and then removed afterwards. That is why you can observe a delay before the new position is sent. The command topic isn’t bridged to ROS 2 by default. It becomes visible to ROS 2 when a publisher node is active, which is the case when running the “`ros2 topic pub`” command. Using a custom C++ or Python node will solve this delay as the bridge will be created once the node is running and be removed once it is stopped.

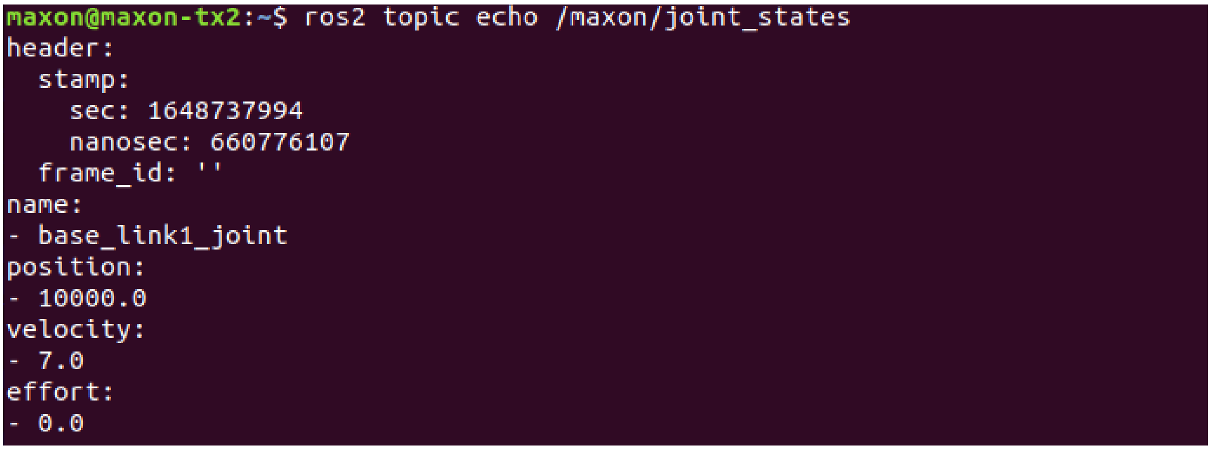
In case you want to display the `JointStates` topic from ROS 1 in a ROS 2 terminal you need to restart the `dynamic_bridge` node with a special option to bridge all the topics (otherwise ROS 2 can’t find and therefore echo the topic). Just press CRTL-C, go up and append the option, which makes this command:

```
$ ros2 run ros1_bridge dynamic_bridge --bridge-all-topics
```

You can then open a new terminal to display the content of the `JointStates` topic using the echo command from ROS 2:

```
$ ros2 topic echo /maxon/joint_states
```

Which should give a display like this:



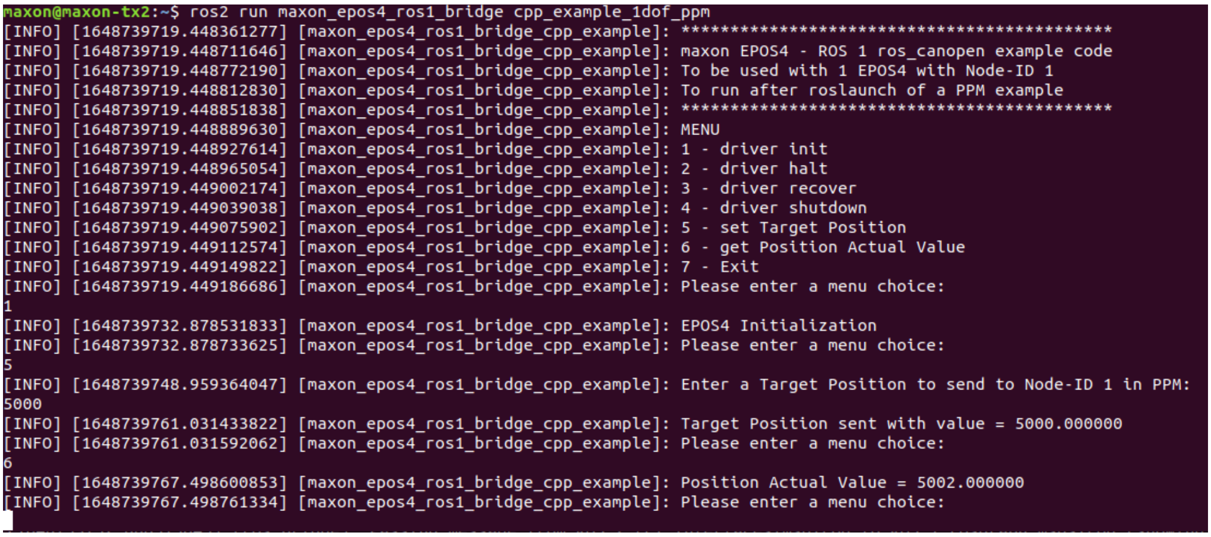
```
maxon@maxon-tx2:~$ ros2 topic echo /maxon/joint_states
header:
  stamp:
    sec: 1648737994
    nanosec: 660776107
  frame_id: ''
name:
- base_link1_joint
position:
- 10000.0
velocity:
- 7.0
effort:
- 0.0
```

5.2. Using the C++ example

After launching the 1DOF PPM launch files and the `dynamic_bridge` as in §5.1, just run the compiled node like this in a new terminal after sourcing ROS 2 as usual:

```
$ source ~/dev_ws/install/setup.bash
$ ros2 run maxon_epos4_ros1_bridge cpp_example_1dof_ppm
```

After the menu appears, follow the instructions on the screen. You can first initialize the driver with choice “1”, then set Target Position with choice “5”, and then read it back with choice “6”:

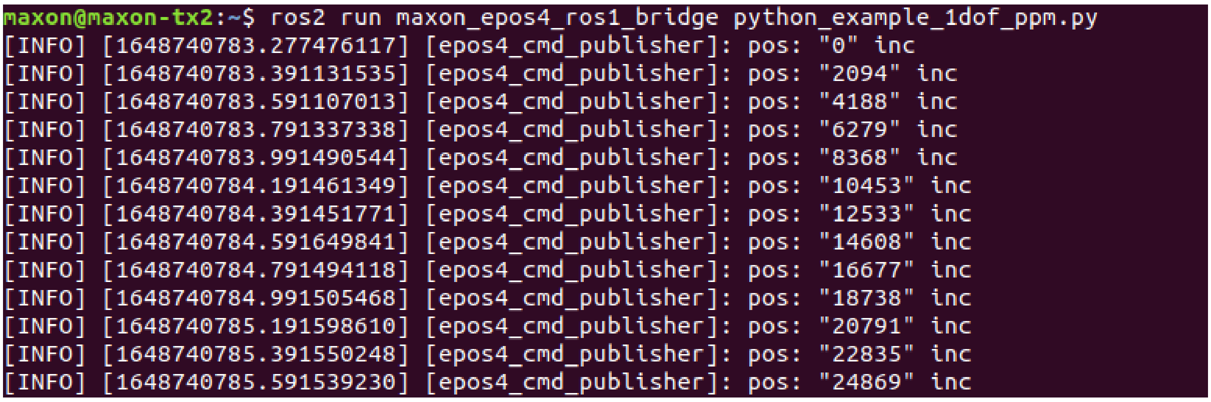


```
maxon@maxon-tx2:~$ ros2 run maxon_epos4_ros1_bridge cpp_example_1dof_ppm
[INFO] [1648739719.448361277] [maxon_epos4_ros1_bridge_cpp_example]: ********************************************
[INFO] [1648739719.448711646] [maxon_epos4_ros1_bridge_cpp_example]: maxon EPOS4 - ROS 1 ros_canopen example code
[INFO] [1648739719.448772190] [maxon_epos4_ros1_bridge_cpp_example]: To be used with 1 EPOS4 with Node-ID 1
[INFO] [1648739719.448812830] [maxon_epos4_ros1_bridge_cpp_example]: To run after roslaunch of a PPM example
[INFO] [1648739719.448851838] [maxon_epos4_ros1_bridge_cpp_example]: ********************************************
[INFO] [1648739719.448889630] [maxon_epos4_ros1_bridge_cpp_example]: MENU
[INFO] [1648739719.448927614] [maxon_epos4_ros1_bridge_cpp_example]: 1 - driver init
[INFO] [1648739719.448965054] [maxon_epos4_ros1_bridge_cpp_example]: 2 - driver halt
[INFO] [1648739719.449002174] [maxon_epos4_ros1_bridge_cpp_example]: 3 - driver recover
[INFO] [1648739719.449039038] [maxon_epos4_ros1_bridge_cpp_example]: 4 - driver shutdown
[INFO] [1648739719.449075902] [maxon_epos4_ros1_bridge_cpp_example]: 5 - set Target Position
[INFO] [1648739719.449112574] [maxon_epos4_ros1_bridge_cpp_example]: 6 - get Position Actual Value
[INFO] [1648739719.449149822] [maxon_epos4_ros1_bridge_cpp_example]: 7 - Exit
[INFO] [1648739719.449186686] [maxon_epos4_ros1_bridge_cpp_example]: Please enter a menu choice:
1
[INFO] [1648739732.878531833] [maxon_epos4_ros1_bridge_cpp_example]: EPOS4 Initialization
[INFO] [1648739732.878733625] [maxon_epos4_ros1_bridge_cpp_example]: Please enter a menu choice:
5
[INFO] [1648739748.959364047] [maxon_epos4_ros1_bridge_cpp_example]: Enter a Target Position to send to Node-ID 1 in PPM:
5000
[INFO] [1648739761.031433822] [maxon_epos4_ros1_bridge_cpp_example]: Target Position sent with value = 5000.000000
[INFO] [1648739761.031592062] [maxon_epos4_ros1_bridge_cpp_example]: Please enter a menu choice:
6
[INFO] [1648739767.498600853] [maxon_epos4_ros1_bridge_cpp_example]: Position Actual Value = 5002.000000
[INFO] [1648739767.498761334] [maxon_epos4_ros1_bridge_cpp_example]: Please enter a menu choice:
```

5.3. Using the Python example

Similarly as in §5.1, execute the 1DOF PPM launch file, run the `dynamic_bridge` and call the `init` service of the driver in different terminals. You can then run the example with “`ros2 run`”:

```
$ ros2 run maxon_epos4_ros1_bridge python_example_1dof_ppm.py
```



```
maxon@maxon-tx2:~$ ros2 run maxon_epos4_ros1_bridge python_example_1dof_ppm.py
[INFO] [1648740783.277476117] [epos4_cmd_publisher]: pos: "0" inc
[INFO] [1648740783.391131535] [epos4_cmd_publisher]: pos: "2094" inc
[INFO] [1648740783.591107013] [epos4_cmd_publisher]: pos: "4188" inc
[INFO] [1648740783.791337338] [epos4_cmd_publisher]: pos: "6279" inc
[INFO] [1648740783.991490544] [epos4_cmd_publisher]: pos: "8368" inc
[INFO] [1648740784.191461349] [epos4_cmd_publisher]: pos: "10453" inc
[INFO] [1648740784.391451771] [epos4_cmd_publisher]: pos: "12533" inc
[INFO] [1648740784.591649841] [epos4_cmd_publisher]: pos: "14608" inc
[INFO] [1648740784.791494118] [epos4_cmd_publisher]: pos: "16677" inc
[INFO] [1648740784.991505468] [epos4_cmd_publisher]: pos: "18738" inc
[INFO] [1648740785.191598610] [epos4_cmd_publisher]: pos: "20791" inc
[INFO] [1648740785.391550248] [epos4_cmd_publisher]: pos: "22835" inc
[INFO] [1648740785.591539230] [epos4_cmd_publisher]: pos: "24869" inc
```