APPLICATION NOTE

```
// Create an instant camera object with the firs
Camera_t camera( CTlFactory::GetInstance().Creat

// Register an image event handler that accesses
camera.RegisterImageEventHandler( new CSampleIma
Ownership_TakeOwnership);

// Open the camera.
camera.Open();
```

Interfacing Basler Cameras with ROS 2

Applicable to Basler camera(s) that allow images to be displayed by the Basler pylon Viewer only

Document Number: AW001729
Version: 01 Language: 000 (English)
Release Date: 15 May 2022

Contacting Basler Support Worldwide

Europe, Middle East, Africa

Basler AG
An der Strusbek 60–62
22926 Ahrensburg
Germany

Tel. +49 4102 463 515
Fax +49 4102 463 599

support.europe@baslerweb.com

The Americas

Basler, Inc.
855 Springdale Drive, Suite 203
Exton, PA 19341
USA

Tel. +1 610 280 0171
Fax +1 610 280 7608

support.usa@baslerweb.com

Asia-Pacific

Basler Asia Pte. Ltd.
35 Marsiling Industrial Estate Road 3
#05–06
Singapore 739257

Tel. +65 6367 1355
Fax +65 6367 1255

support.asia@baslerweb.com

www.baslerweb.com

Table of Contents

1 General Information

This application note describes how to interface Basler GigE and USB3 Vision cameras with ROS 2 using the pylon-ros2-camera driver package (expressed in code as `pylon_ros2_camera`).

Sensors and cameras are commonly used in robotics. The sensors are single-information and array detectors while cameras provide visual control. To interface cameras for robotics the Robot Operating System (ROS) user community continues to create camera driver wrappers and processing nodes.

ROS is an all open-source framework of software libraries and tools. The framework supports the building of various robot applications. ROS provides the developing tools, algorithms and drivers for a variety of robotics platform projects.

ROS can run a large number of executables (nodes) in parallel and allows them to exchange data synchronously (service) or asynchronously (subscribed/published topics). In practice, the data are generally sensor queries whose result data are processed to cause robot actions.

Since ROS was started, a lot has changed in the robotics and ROS community. The aim of the latest ROS 2 project is to adapt to these changes, leveraging what is positive about ROS and improving what isn't yet.

The procedures described in this document were evaluated with Basler pylon v. 6.3 installed and with the following Linux distribution and ROS software:

- Ubuntu 20.04.4 LTS (Focal Fossa) 64-bit
- ROS 2 (Galactic Geochalone)

Check pylon version compatibilities when creating or using further ROS 2 nodes.

This document shows command examples after the $ prompt. You can use them via copy-and-paste.

Legal Notice

Basler does not assume any liability for the functionality and suitability of any recommended open-source products referenced in this application note. This is just a presentation of a sample use case. The readers of this application note are fully responsible to conduct their own testing procedures to assess the suitability of the mentioned open-source products for their own applications.

The procedures described in this document assume that you are using the following hardware components and software:

- Linux x86_64 operating system
- A Basler 2D GigE or USB 3.0 camera

- pylon Viewer version 6.2 or newer
- pylon-ros2-camera driver package
- ROS 2 Robot Operating System

2 Installation

The installation section describes how to install the following software:

- Operating system
- Basler pylon Camera Software Suite for Linux x86_64
- ROS 2 Robot Operating System
- pylon-ros2-camera driver package

2.1 Operating System Compatibilities

This document focuses on the ROS 2 use with natively installed Linux x86_64 operating systems and assumes that you use or create a new operating system installation using a Linux ISO image. In the present case an Ubuntu 20.04.4 Long Term Support (LTS) x64 installation has been used. Make sure that an internet connection on your Linux machine is available. In case of any difficulties, check if any proxy server settings are necessary or must be adjusted. If the installations take place behind a proxy server, at least proper HTTPS and FTP settings including port access are mandatory.

Since ROS 2 is officially compatible with Windows 10 operating system, the pylon-ros2-camera driver package may be as well. However, such constellations have never been tried, let alone tested.

2.2 Installing the Basler pylon Camera Software Suite for Linux x86_64

The pylon-ros2-camera driver package requires that the library of pylon version 6.2 or newer is installed. At the moment, a manual installation is required especially with the latest version. The following situations can apply:

- pylon is already installed and path variable `PYLON_ROOT` is set properly
- pylon is not yet installed but will be manually installed and enabled to be applicable for ROS 2 nodes

If you need to install a suitable pylon version, continue with the following steps. Otherwise, continue with chapter 2.3 Installing the ROS 2 Robot Operating System further below.

To install pylon Viewer version 6.2 or higher:

1. Go to https://www.baslerweb.com/en/downloads/software-downloads/ where two pylon Camera Software Suites for Linux x86_64 installer packages are available.
2. Download one of both packages, depending on applicability:

- **tar.gz** (applicable to all Linux distributions)
- **.deb** (applicable to Ubuntu and related Linux distributions)

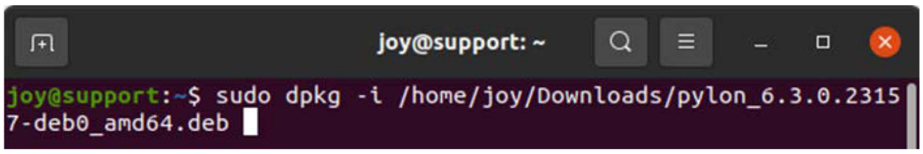
3. Install the downloaded installer package.
   - If you downloaded **tar.gz**:

   a. Install the pylon SDK from the **tar.gz** installer package. Details about installation and configuration are available from the included **INSTALL** and **README** files.

ℹ	Make sure to carry out the necessary adjustments as described in the **INSTALL** file: ▪ Run the **pylon-setup-env.sh** script to set the `PYLON_ROOT` environment variable. ▪ If you want to use Basler USB3 Vision cameras, run the included **setup-usb.sh** script.

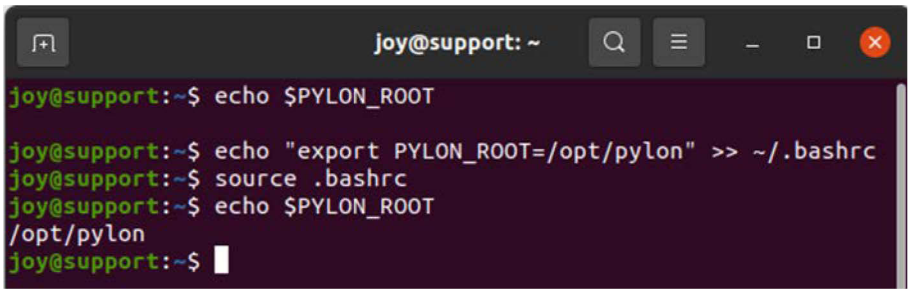
   - If you downloaded **.deb**:

   a. Install the pylon SDK for Linux on Debian and related Linux distributions (e.g., Ubuntu) from the **.deb** installer package that suits your platform. To do so, open the `dpkg` command line tool:



```
joy@support:~$ sudo dpkg -i /home/joy/Downloads/pylon_6.3.0.2315
7-deb0_amd64.deb
```

   b. Check the `pylon root` location environment variable and make sure it exists. If not, type the following:
   `$ echo “export PYLON_ROOT=/opt/pylon” >> ~/.bashrc` variable creation to the **~/.bashrc** file.



```
joy@support:~$ echo $PYLON_ROOT

joy@support:~$ echo "export PYLON_ROOT=/opt/pylon" >> ~/.bashrc
joy@support:~$ source .bashrc
joy@support:~$ echo $PYLON_ROOT
/opt/pylon
joy@support:~$ 
```

The `PYLON_ROOT` environment variable is necessary for pylon path identification related to development and pylon-ros2-camera driver package use. See below for more information about pylon-ros2-camera, designed for use with cameras supported by pylon.

2.3 Installing the ROS 2 Robot Operating System

The following installation steps are listed without detailed comment. For additional information, see the ROS 2 Documentation.

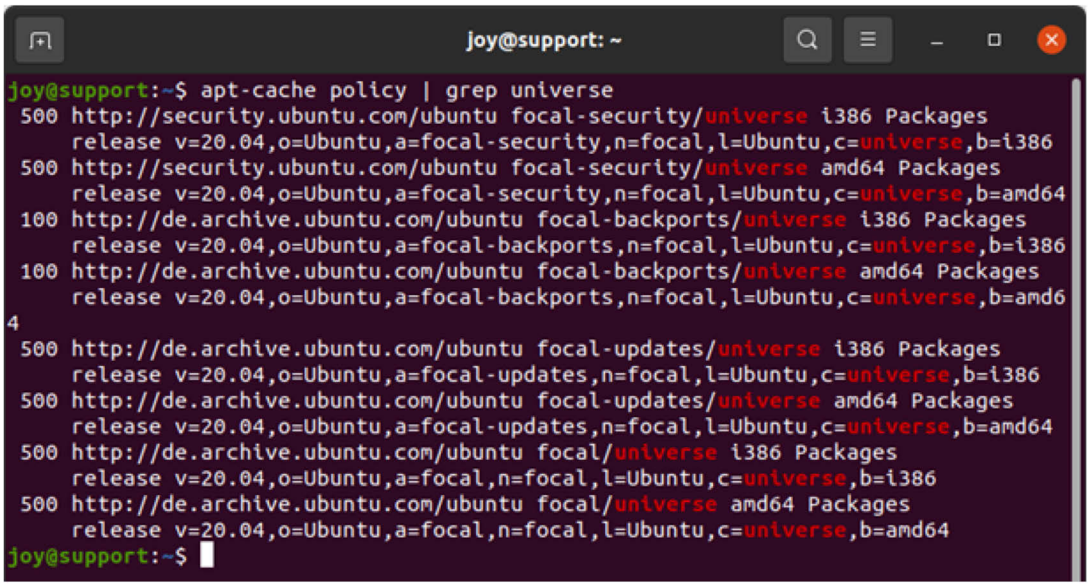
Below, the installation of ROS 2 Galactic Geochelone is described. It is the currently released version. For more details and possible alternative installation steps visit the Ubuntu ROS 2 Galactic Geochelone Installation site.

This application note may also apply to other ROS 2 releases, with installations analogous to the installation of ROS 2 Galactic Geochelone. This, however, was not tested.

To install ROS 2 Galactic Geochelone:

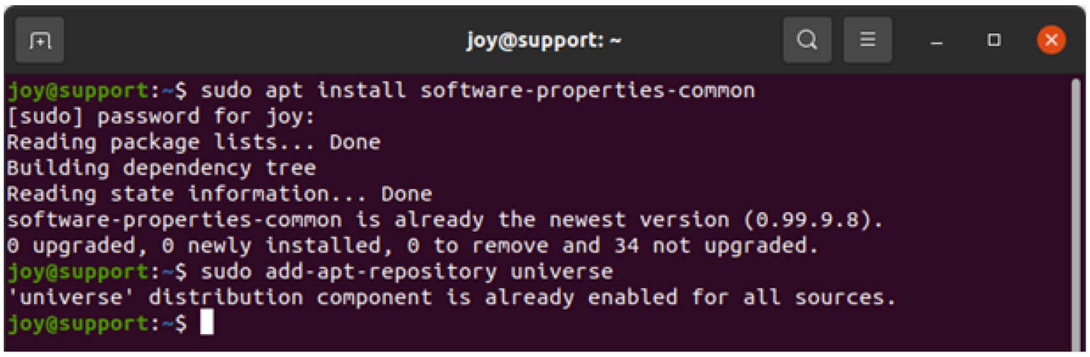
1. Prepare the installation with adding the ROS 2 apt repository to the system.
2. In the dpgk command line tool, check if the Ubuntu Universe repository is enabled by typing the following commands:

   ```
   $ apt-cache policy | grep universe
   ```



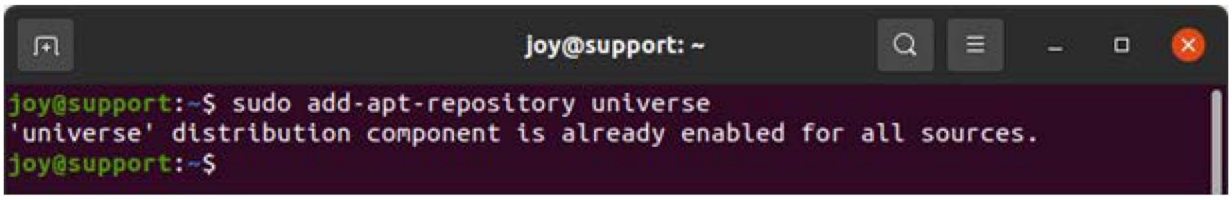
```
joy@support:~$ apt-cache policy | grep universe
 500 http://security.ubuntu.com/ubuntu focal-security/universe i386 Packages
     release v=20.04,o=Ubuntu,a=focal-security,n=focal,l=Ubuntu,c=universe,b=i386
 500 http://security.ubuntu.com/ubuntu focal-security/universe amd64 Packages
     release v=20.04,o=Ubuntu,a=focal-security,n=focal,l=Ubuntu,c=universe,b=amd64
 100 http://de.archive.ubuntu.com/ubuntu focal-backports/universe i386 Packages
     release v=20.04,o=Ubuntu,a=focal-backports,n=focal,l=Ubuntu,c=universe,b=i386
 100 http://de.archive.ubuntu.com/ubuntu focal-backports/universe amd64 Packages
     release v=20.04,o=Ubuntu,a=focal-backports,n=focal,l=Ubuntu,c=universe,b=amd6
4
 500 http://de.archive.ubuntu.com/ubuntu focal-updates/universe i386 Packages
     release v=20.04,o=Ubuntu,a=focal-updates,n=focal,l=Ubuntu,c=universe,b=i386
 500 http://de.archive.ubuntu.com/ubuntu focal-updates/universe amd64 Packages
     release v=20.04,o=Ubuntu,a=focal-updates,n=focal,l=Ubuntu,c=universe,b=amd64
 500 http://de.archive.ubuntu.com/ubuntu focal/universe i386 Packages
     release v=20.04,o=Ubuntu,a=focal,n=focal,l=Ubuntu,c=universe,b=i386
 500 http://de.archive.ubuntu.com/ubuntu focal/universe amd64 Packages
     release v=20.04,o=Ubuntu,a=focal,n=focal,l=Ubuntu,c=universe,b=amd64
joy@support:~$
```

```
$ sudo apt install software-properties-common
```



```
joy@support:~$ sudo apt install software-properties-common
[sudo] password for joy:
Reading package lists... Done
Building dependency tree
Reading state information... Done
software-properties-common is already the newest version (0.99.9.8).
0 upgraded, 0 newly installed, 0 to remove and 34 not upgraded.
joy@support:~$ sudo add-apt-repository universe
'universe' distribution component is already enabled for all sources.
joy@support:~$
```

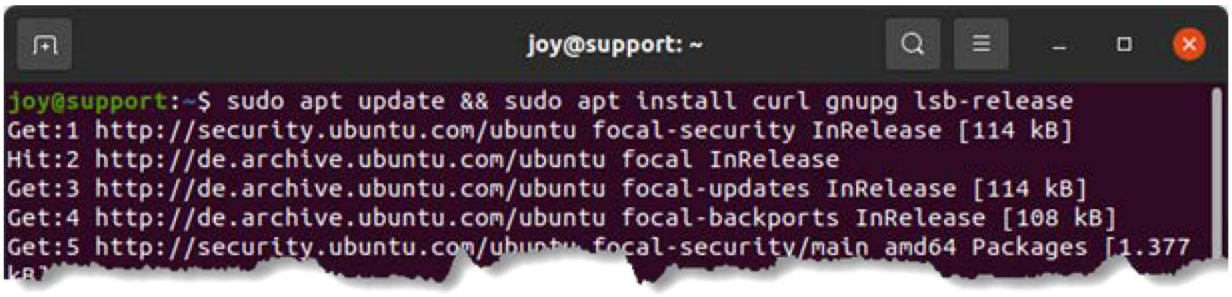
```
$ sudo add-apt-repository universe
```



```
joy@support:~$ sudo add-apt-repository universe
'universe' distribution component is already enabled for all sources.
joy@support:~$
```

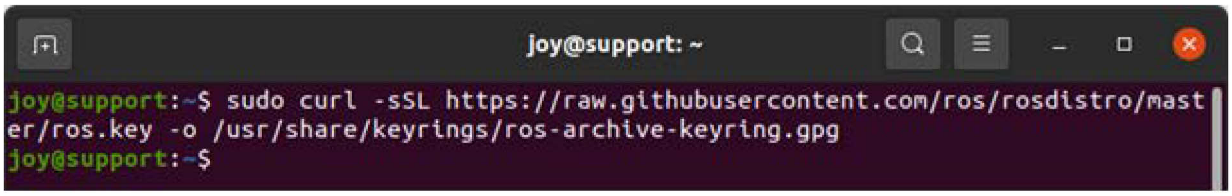
3. Add the ROS 2 apt repository to the system and sources list by typing the following commands:

   ```
   $ sudo apt update && sudo apt install curl gnupg lsb-release
   ```



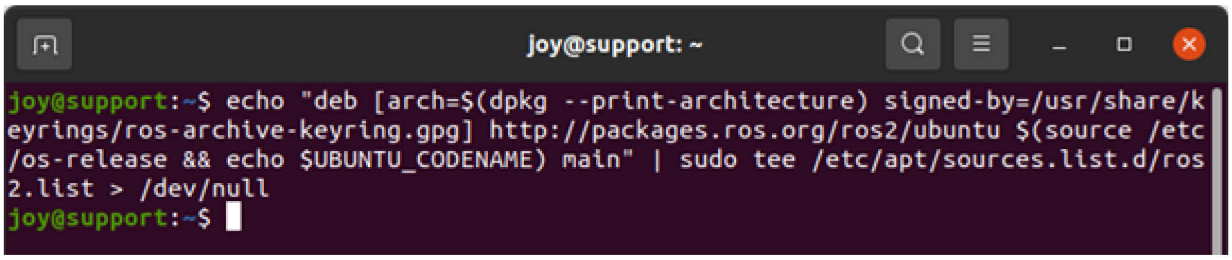
```
joy@support:~$ sudo apt update && sudo apt install curl gnupg lsb-release
Get:1 http://security.ubuntu.com/ubuntu focal-security InRelease [114 kB]
Hit:2 http://de.archive.ubuntu.com/ubuntu focal InRelease
Get:3 http://de.archive.ubuntu.com/ubuntu focal-updates InRelease [114 kB]
Get:4 http://de.archive.ubuntu.com/ubuntu focal-backports InRelease [108 kB]
Get:5 http://security.ubuntu.com/ubuntu focal-security/main amd64 Packages [1.377
```

```
$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key -o
/usr/share/keyrings/ros-archive-keyring.gpg
```



```
joy@support:~$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/mast
er/ros.key -o /usr/share/keyrings/ros-archive-keyring.gpg
joy@support:~$
```

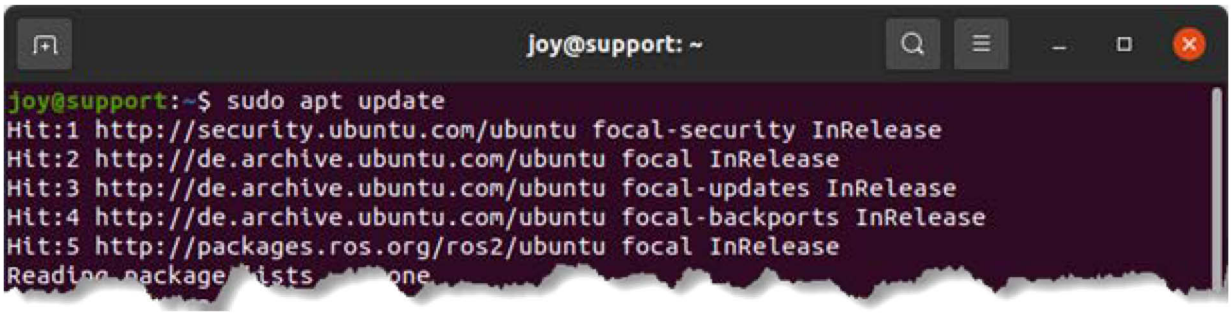
```
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-
keyring.gpg] http://packages.ros.org/ros2/ubuntu $(source /etc/os-release && echo
$UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```



```
joy@support:~$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/k
eyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(source /etc
/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros
2.list > /dev/null
joy@support:~$
```

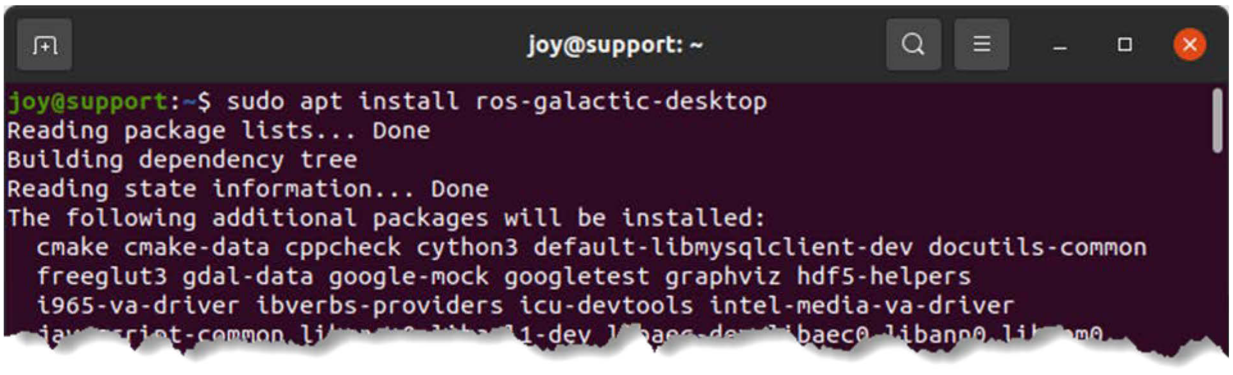
4. Install ROS 2 Galactic Geochelone by typing the following commands:

```
$ sudo apt-get update
```



```
joy@support:~$ sudo apt update
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:2 http://de.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://de.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://de.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:5 http://packages.ros.org/ros2/ubuntu focal InRelease
Reading package lists... Done
```

```
$ sudo apt install ros-galactic-desktop
```



```
joy@support:~$ sudo apt install ros-galactic-desktop
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:
  cmake cmake-data cppcheck cython3 default-libmysqlclient-dev docutils-common
  freeglut3 gdal-data google-mock googletest graphviz hdf5-helpers
  i965-va-driver ibverbs-providers icu-devtools intel-media-va-driver
```

2.3.1 Environment Settings

2.3.1.1 In Preparation for Use Source the Setup Files of Environment Settings

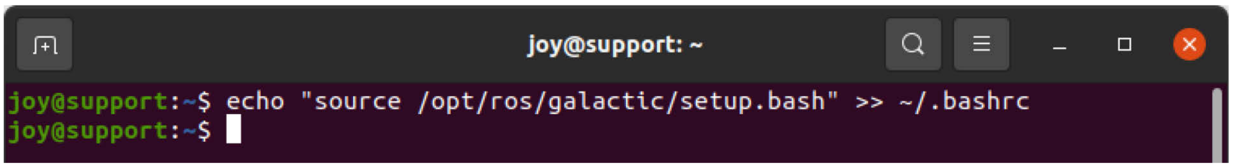
ROS 2 relies on ‘Workspaces’ that are system locations where the developing takes place. Those workspaces are combined for easier developing against different versions. It is accomplished by sourcing setup files every time when opening a new shell. Without that sourcing, which makes packages available, ROS 2 commands are not accessible from the actual shell. In other words, the sourcing allows the shell to know where it has to look to execute ROS commands.

To source the setup files, type the following command:

```
$ source /opt/ros/galactic/setup.bash
```

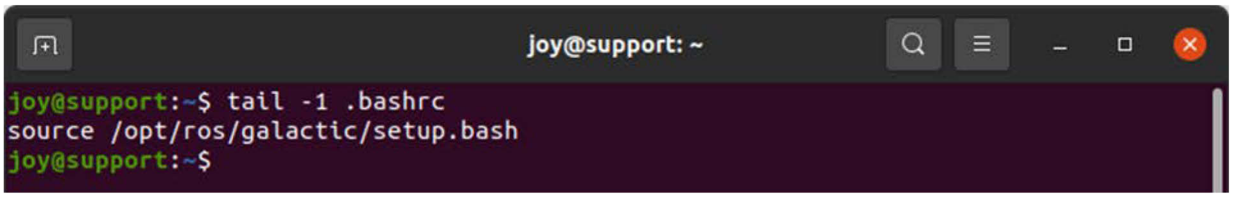
2.3.1.2 Alternative Permanent Setup of Environment Settings

1. If the sourcing of the above setup files is not required every time a new shell is opened, add the following command to the shell startup script:
   `$ echo “source /opt/ros/galactic/setup.bash” >> ~/.bashrc`



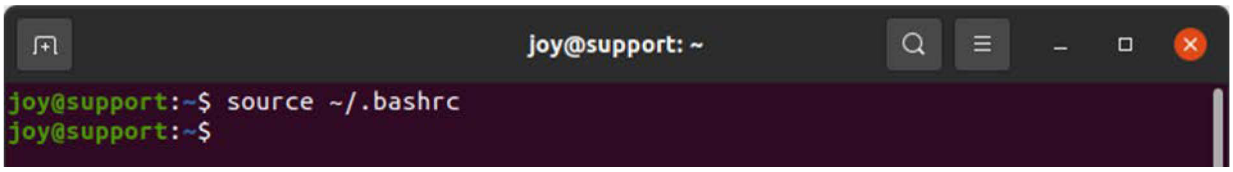
```
joy@support:~$ echo "source /opt/ros/galactic/setup.bash" >> ~/.bashrc
joy@support:~$
```

2. In the shell output, check the correct settings:



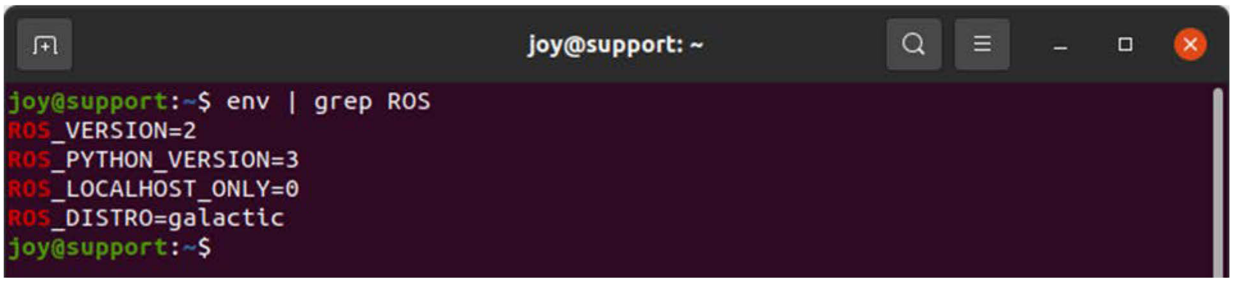
```
joy@support:~$ tail -1 .bashrc
source /opt/ros/galactic/setup.bash
joy@support:~$
```

3. Source the **.bashrc** file to apply the modification with `$ source ~/.bashrc`.



```
joy@support:~$ source ~/.bashrc
joy@support:~$
```

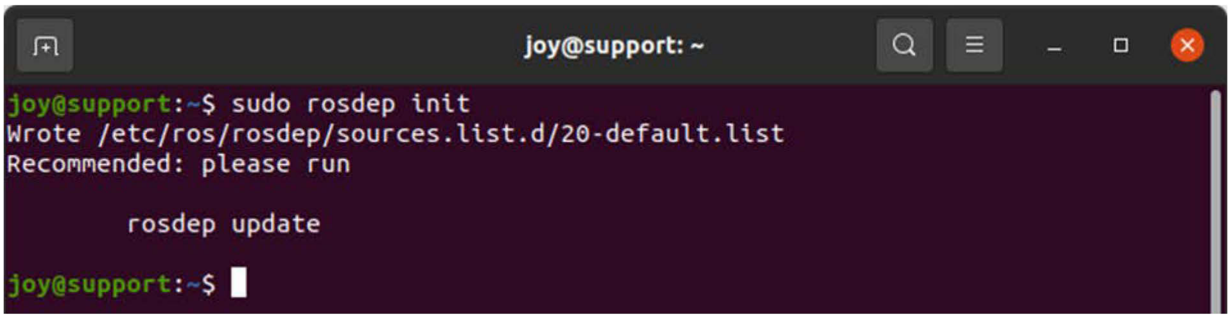
4. Check whether the ROS environment variables were successfully set:



```
joy@support:~$ env | grep ROS
ROS_VERSION=2
ROS_PYTHON_VERSION=3
ROS_LOCALHOST_ONLY=0
ROS_DISTRO=galactic
joy@support:~$
```

2.3.2 Initializing rosdep

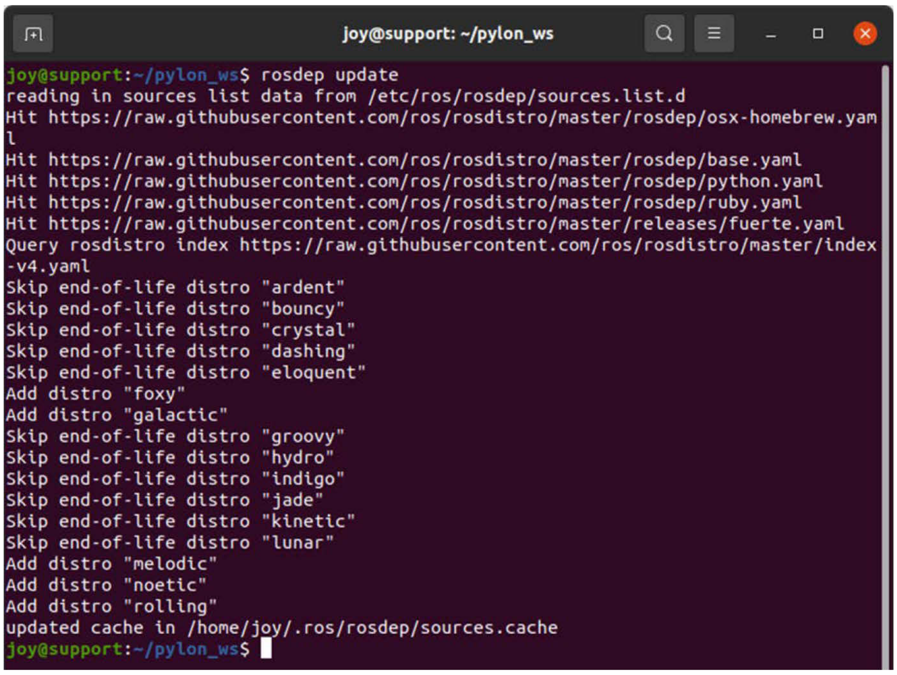
The rosdep init command, `$ sudo rosdep init`, will create a file of dependencies in `/etc/ros/rosdep/sources.list.d` that hold some basic distro dependencies.



```
joy@support:~$ sudo rosdep init
Wrote /etc/ros/rosdep/sources.list.d/20-default.list
Recommended: please run

        rosdep update

joy@support:~$
```

The rosdep update, `$ rosdep update`, will read the distro file mappings and update the information within ROS.

NOTICE
Running a **rosdep** Update with sudo will later result in permission errors. Do not run a **rosdep** Update with sudo.



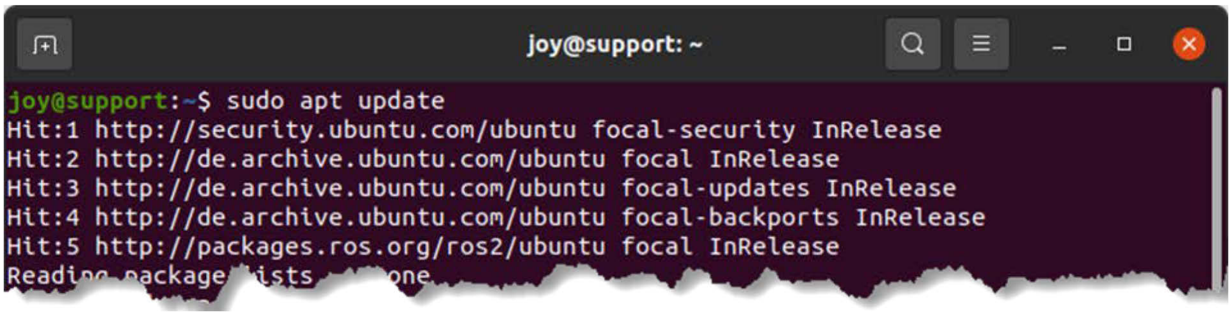
```
joy@support:~/pylon_ws$ rosdep update
reading in sources list data from /etc/ros/rosdep/sources.list.d
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/osx-homebrew.yam
l
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/base.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/python.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/rosdep/ruby.yaml
Hit https://raw.githubusercontent.com/ros/rosdistro/master/releases/fuerte.yaml
Query rosdistro index https://raw.githubusercontent.com/ros/rosdistro/master/index
-v4.yaml
Skip end-of-life distro "ardent"
Skip end-of-life distro "bouncy"
Skip end-of-life distro "crystal"
Skip end-of-life distro "dashing"
Skip end-of-life distro "eloquent"
Add distro "foxy"
Add distro "galactic"
Skip end-of-life distro "groovy"
Skip end-of-life distro "hydro"
Skip end-of-life distro "indigo"
Skip end-of-life distro "jade"
Skip end-of-life distro "kinetic"
Skip end-of-life distro "lunar"
Add distro "melodic"
Add distro "noetic"
Add distro "rolling"
updated cache in /home/joy/.ros/rosdep/sources.cache
joy@support:~/pylon_ws$
```

2.3.3 Installing the Build Tools

A universal tool that automates the process of building packages in their topological order and handles the workflow of environment setup while building and afterwards is called 'colcon'. It must be installed before working with workspaces. It's an interaction of known build tools.

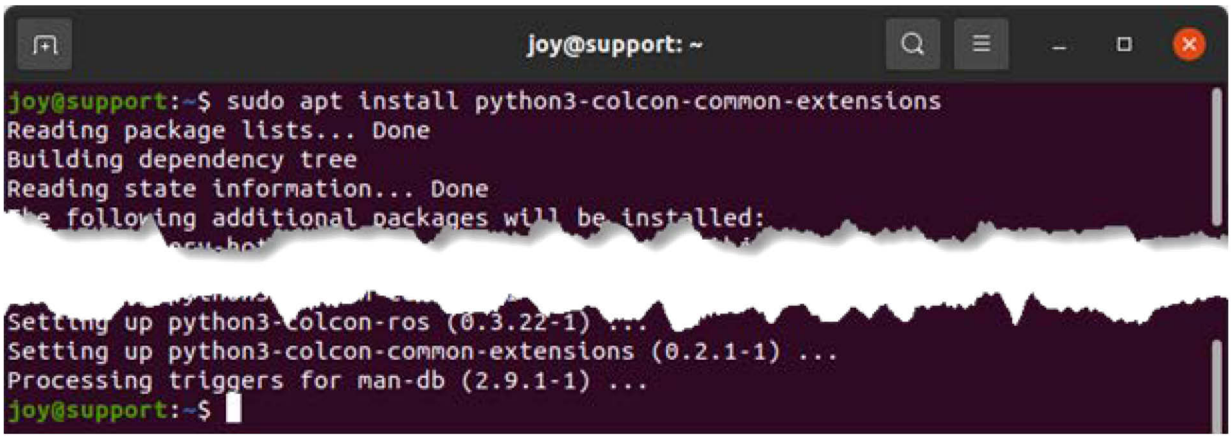
To install the build tool colcon:

1. Type `$ sudo apt update`.



```
joy@support:~$ sudo apt update
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:2 http://de.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://de.archive.ubuntu.com/ubuntu focal-updates InRelease
Hit:4 http://de.archive.ubuntu.com/ubuntu focal-backports InRelease
Hit:5 http://packages.ros.org/ros2/ubuntu focal InRelease
Reading package lists one
```

2. Type `$ sudo apt install python3-colcon-common-extensions`.



```
joy@support:~$ sudo apt install python3-colcon-common-extensions
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following additional packages will be installed:

Setting up python3-colcon-ros (0.3.22-1) ...
Setting up python3-colcon-common-extensions (0.2.1-1) ...
Processing triggers for man-db (2.9.1-1) ...
joy@support:~$
```

Alternative Installation

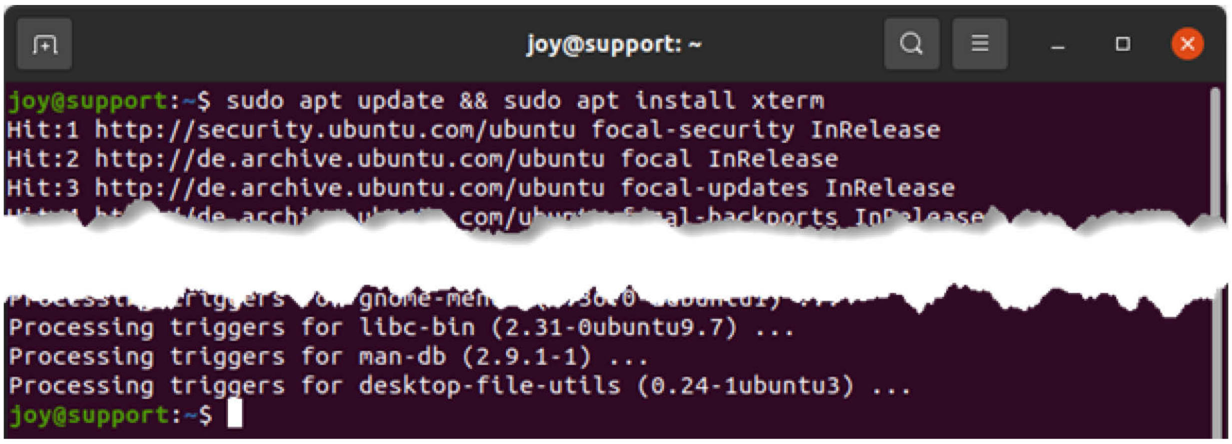
```
<sudo apt install python3-pip>
<pip install –U colcon-common-extensions>
```

2.3.4 Installing Tools

With the ROS 2 launch there are known issues that require the xterm terminal emulator installation and usage so that the stdin user interaction is possible, i.e., with GDB.

To install the xterm terminal emulator:

1. Type `$ sudo apt get update && sudo apt install xterm`.



```
joy@support:~$ sudo apt update && sudo apt install xterm
Hit:1 http://security.ubuntu.com/ubuntu focal-security InRelease
Hit:2 http://de.archive.ubuntu.com/ubuntu focal InRelease
Hit:3 http://de.archive.ubuntu.com/ubuntu focal-updates InRelease

Processing triggers for libc-bin (2.31-0ubuntu9.7) ...
Processing triggers for man-db (2.9.1-1) ...
Processing triggers for desktop-file-utils (0.24-1ubuntu3) ...
joy@support:~$
```

2.4 Installing the Middleware

The descriptions given so far do not consider the intermediary (“driver”) between the powerful pylon and ROS software structures. Such driver is usually created by the ROS-oriented developers community.

The installation of a driver is illustrated here using the pylon-ros2-camera driver package as the driver. In this document it is the pylon_ros2_camera driver of branch ‘galactic’. The installation assumes that operating system and ROS 2 Robot Operating System have already been installed, as described above.

2.4.1 Details About the pylon-ros2-camera driver package

The pylon-ros2-camera driver package is the currently official pylon ROS driver for all recent Basler GigE Vision and USB3 Vision cameras. You can download the driver package using this URL: https://github.com/basler/pylon-ros-camera/archive/refs/heads/galactic.zip.

The driver package provides a range of the pylon API features that allow interactive camera operation. Images are published into ROS. The package is designed to meet certain application tasks and is therefore not a complete wrapper for all pylon API methods. However, adhering to the open-source concept, the pylon-ros2-camera can be studied, copied or modified, observing the related copyright and the BSD license model.

For further information about pylon-ros2-camera, go to its GitHub: GitHub - basler/pylon-ros-camera at galactic.

2.4.2 Preparing a ROS 2 Build Workspace

First off ROS 2 was installed. Then the colcon tools were added. That is a workspace build system and provides low level build system macros and infrastructure. The colcon system is necessary to build code projects like pylon-ros2-camera, for example.

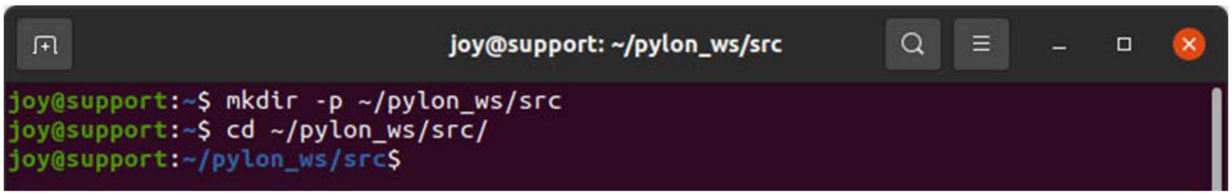
2.4.2.1 Creating a Working Directory

A workspace must be set up where single or multiple packages can be built. While the directory name can be chosen freely, it is advisable to link it to the purpose of the workspace, i.e. **pylon_ws**.

In the following, the folder **pylon_ws** and its subfolder **src** are created, unless they are already present.

To create folders:

1. Type `$ mkdir -p ~/pylon_ws/src` and `$ cd ~/pylon_ws/src`.

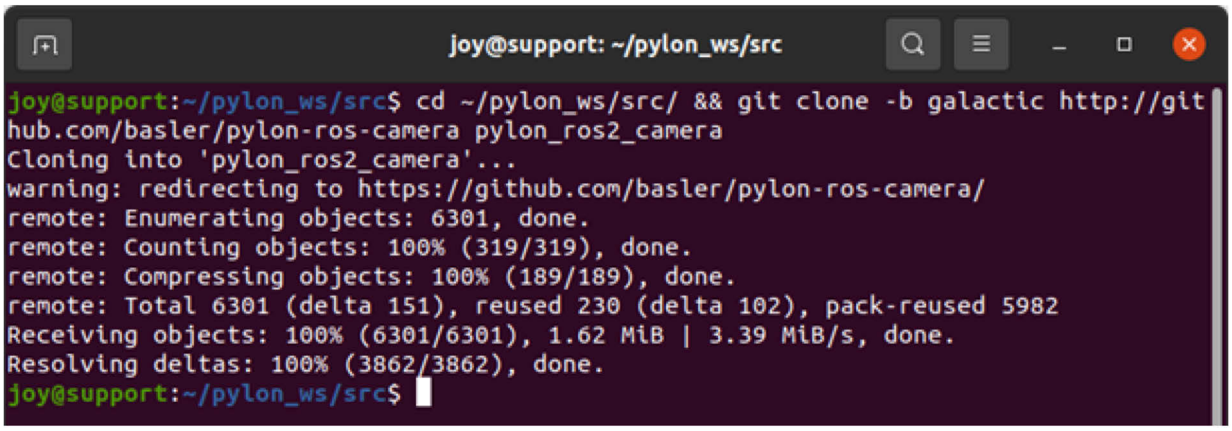


```
joy@support: ~/pylon_ws/src
joy@support:~$ mkdir -p ~/pylon_ws/src
joy@support:~$ cd ~/pylon_ws/src/
joy@support:~/pylon_ws/src$
```

Later on in the process, the ROS 2 packages are cloned into the **src** folder for building. Creating a new directory for any new workspace is a good practice as well as placing packages within a **src** subdirectory.

2.4.3 The Driver Employment

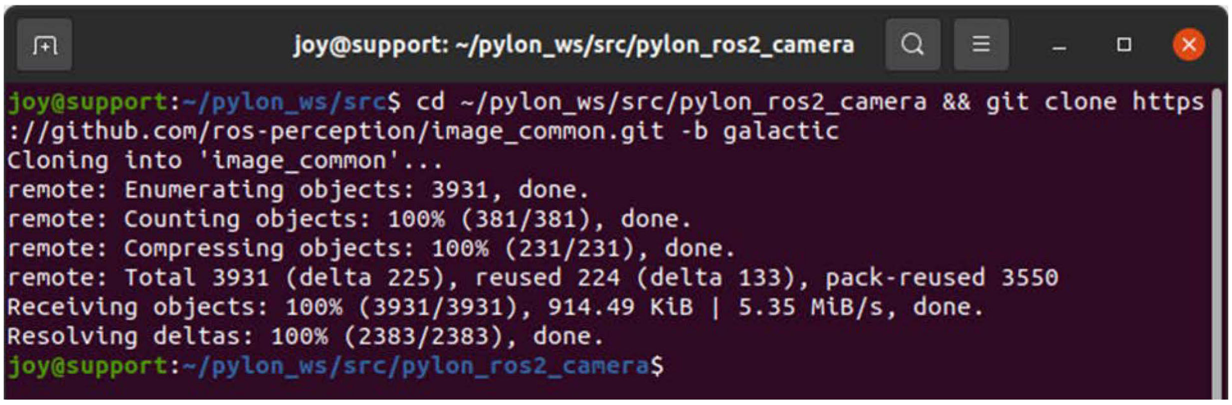
1. Clone the necessary driver packages from GitHub to the related colcon build system workspace **src** folder: `$ cd ~/pylon_ws/src/ && git clone -b galactic https://github.com/basler/pylon-ros-camera pylon_ros2_camera`



```
joy@support:~/pylon_ws/src$ cd ~/pylon_ws/src/ && git clone -b galactic http://git
hub.com/basler/pylon-ros-camera pylon_ros2_camera
Cloning into 'pylon_ros2_camera'...
warning: redirecting to https://github.com/basler/pylon-ros-camera/
remote: Enumerating objects: 6301, done.
remote: Counting objects: 100% (319/319), done.
remote: Compressing objects: 100% (189/189), done.
remote: Total 6301 (delta 151), reused 230 (delta 102), pack-reused 5982
Receiving objects: 100% (6301/6301), 1.62 MiB | 3.39 MiB/s, done.
Resolving deltas: 100% (3862/3862), done.
joy@support:~/pylon_ws/src$
```

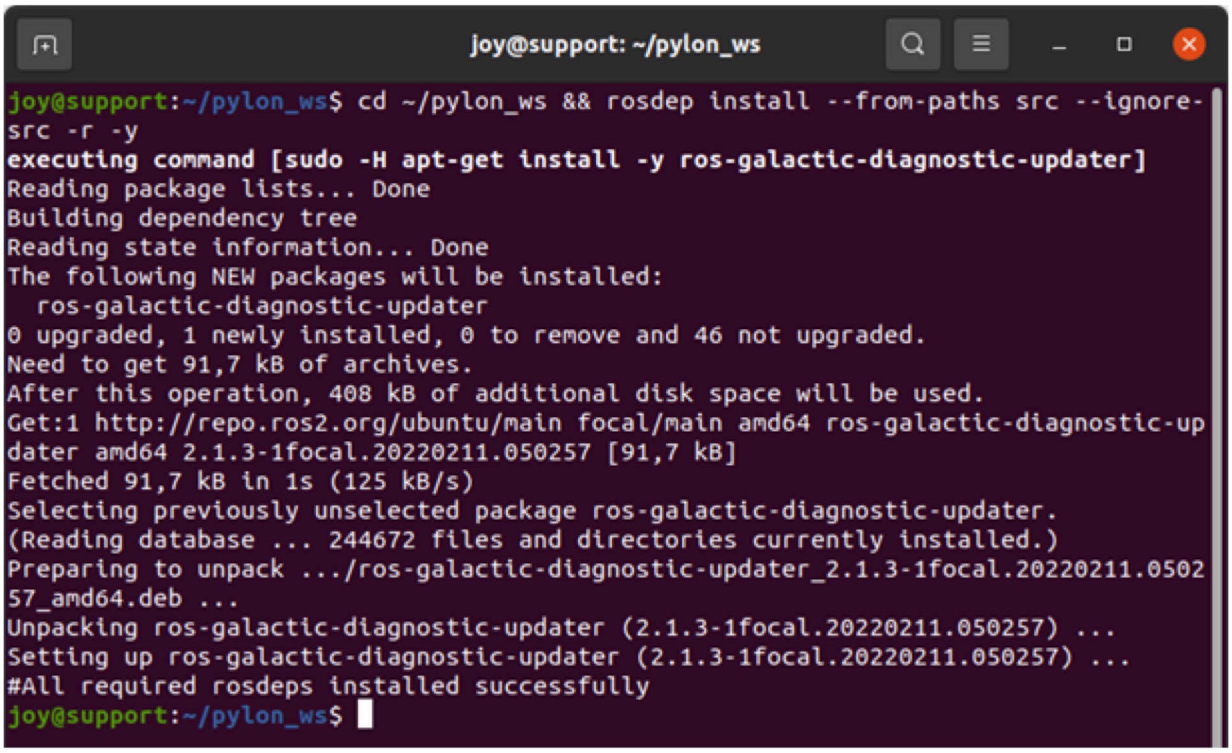
Due to a recent issue with ROS CameraPublisher::getNumSubscribers it's not possible to count the correct number of subscribers to the image_raw and image_rect topics. For a fix it is required to clone the image_common package as well. It will be compiled with the pylon_ros2_camera_node.

2. Clone the necessary additional package from GitHub to the related colcon build system workspace **src/pylon_ros2_camera** folder:
`$ cd ~/pylon_ws/src/pylon_ros2_camera && git clone -b galactic https://github.com/ros-perception/image.common.git pylon_ros2_camera`



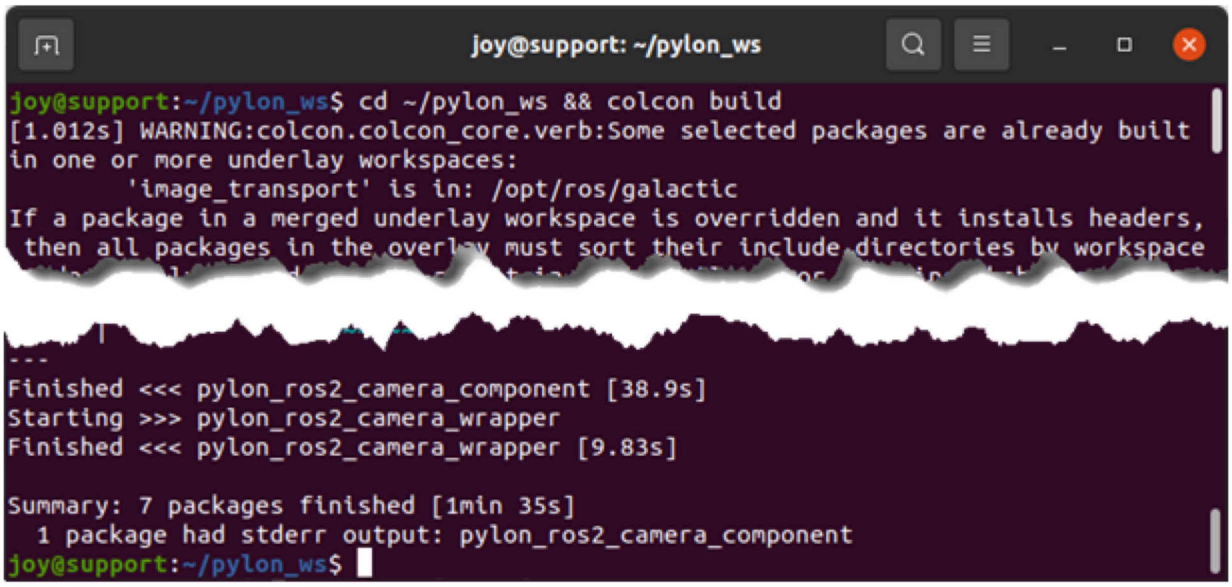
```
joy@support:~/pylon_ws/src$ cd ~/pylon_ws/src/pylon_ros2_camera && git clone https
://github.com/ros-perception/image_common.git -b galactic
Cloning into 'image_common'...
remote: Enumerating objects: 3931, done.
remote: Counting objects: 100% (381/381), done.
remote: Compressing objects: 100% (231/231), done.
remote: Total 3931 (delta 225), reused 224 (delta 133), pack-reused 3550
Receiving objects: 100% (3931/3931), 914.49 KiB | 5.35 MiB/s, done.
Resolving deltas: 100% (2383/2383), done.
joy@support:~/pylon_ws/src/pylon_ros2_camera$
```

3. Install mandatory dependencies by typing `$ cd ~/pylon_ws && sudo rosdep install --from-paths src --ignore-src -r -y`.



```
joy@support: ~/pylon_ws
joy@support:~/pylon_ws$ cd ~/pylon_ws && rosdep install --from-paths src --ignore-
src -r -y
executing command [sudo -H apt-get install -y ros-galactic-diagnostic-updater]
Reading package lists... Done
Building dependency tree
Reading state information... Done
The following NEW packages will be installed:
  ros-galactic-diagnostic-updater
0 upgraded, 1 newly installed, 0 to remove and 46 not upgraded.
Need to get 91,7 kB of archives.
After this operation, 408 kB of additional disk space will be used.
Get:1 http://repo.ros2.org/ubuntu/main focal/main amd64 ros-galactic-diagnostic-up
dater amd64 2.1.3-1focal.20220211.050257 [91,7 kB]
Fetched 91,7 kB in 1s (125 kB/s)
Selecting previously unselected package ros-galactic-diagnostic-updater.
(Reading database ... 244672 files and directories currently installed.)
Preparing to unpack .../ros-galactic-diagnostic-updater_2.1.3-1focal.20220211.0502
57_amd64.deb ...
Unpacking ros-galactic-diagnostic-updater (2.1.3-1focal.20220211.050257) ...
Setting up ros-galactic-diagnostic-updater (2.1.3-1focal.20220211.050257) ...
#All required rosdeps installed successfully
joy@support:~/pylon_ws$
```

4. Change to the workspace folder.
5. Build **pylon_ros2_camera** using **colcon build** by typing `$ cd ~/pylon_ws && colcon build`.



```
joy@support: ~/pylon_ws
joy@support:~/pylon_ws$ cd ~/pylon_ws && colcon build
[1.012s] WARNING:colcon.colcon_core.verb:Some selected packages are already built
in one or more underlay workspaces:
        'image_transport' is in: /opt/ros/galactic
If a package in a merged underlay workspace is overridden and it installs headers,
 then all packages in the overlay must sort their include directories by workspace

---
Finished <<< pylon_ros2_camera_component [38.9s]
Starting >>> pylon_ros2_camera_wrapper
Finished <<< pylon_ros2_camera_wrapper [9.83s]

Summary: 7 packages finished [1min 35s]
  1 package had stderr output: pylon_ros2_camera_component
joy@support:~/pylon_ws$
```

i	Since packages are built in type RELEASE it has to be used an argument to build them in type DEBUG, i.e. `colcon build --symlink-install --cmake-args=-DCMAKE_BUILD_TYPE=Debug`

2.4.3.1 In Preparation for Use Source the Setup Files of Workspace Settings

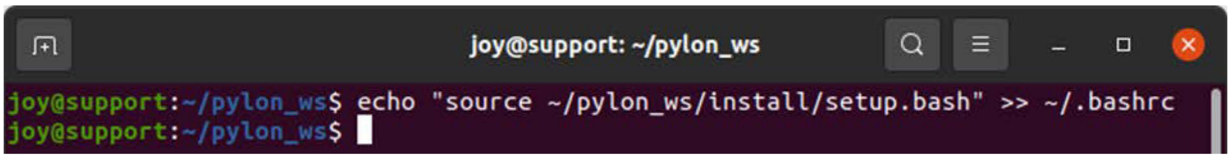
When you open a new shell, it searches for commands only in certain areas of the entire file system. In other words, when you type a command, it will search in some predefined areas to find out if it is an actual command.
Now, if required to add new commands to your shell, it has to know where to find them. This is basically the workspace **setup.bash** file telling, where to find all the ROS executables (including the ones compiled).

To add new commands to your shell (if required):

1. Type `$ source ~/pylon_ws/install /setup.bash`

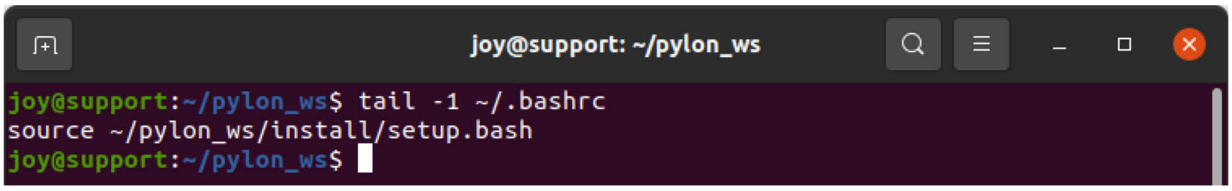
2.4.3.2 Alternative Permanent Setup of Environment Settings

If the sourcing of setup files is not required every time a new shell is opened, add the following command to the shell startup script: `$ echo “source ~/pylon_ws/install/setup.bash” >> ~/.bashrc`.



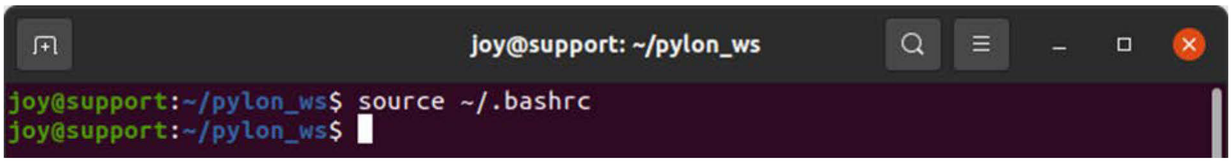
```
joy@support:~/pylon_ws$ echo "source ~/pylon_ws/install/setup.bash" >> ~/.bashrc
joy@support:~/pylon_ws$
```

1. In the shell output, check the correct settings:



```
joy@support:~/pylon_ws$ tail -1 ~/.bashrc
source ~/pylon_ws/install/setup.bash
joy@support:~/pylon_ws$
```

2. Source the **.bashrc** file to apply the modification with `$ source ~/.bashrc`.



```
joy@support:~/pylon_ws$ source ~/.bashrc
joy@support:~/pylon_ws$
```

2.4.4 Running the Driver Package

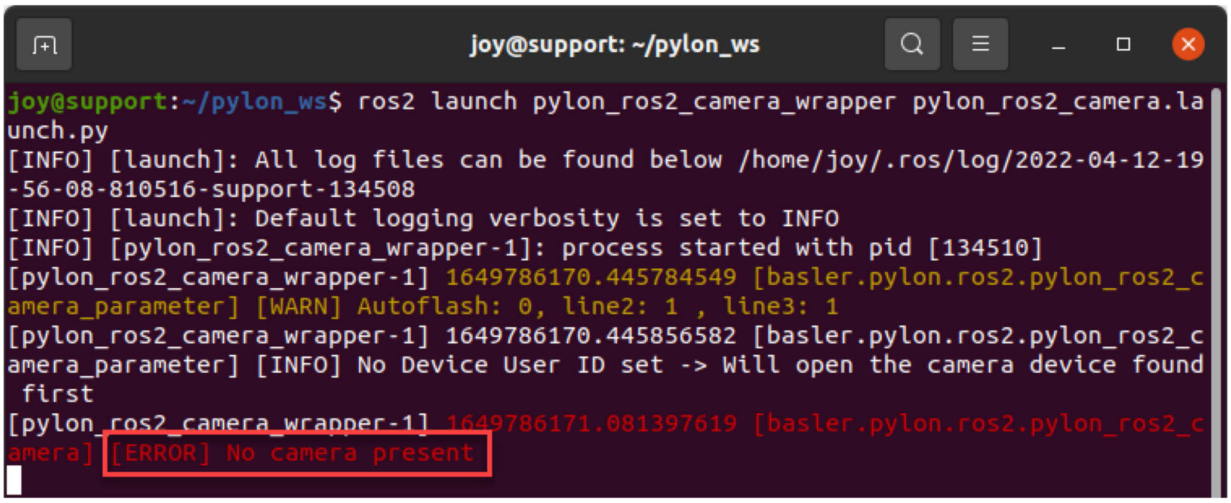
To run the driver package:

1. Type `$ ros2 launch pylon_ros2_camera_wrapper pylon_ros2_camera.launch.py`

This automatically uses the first camera model that is found by underlaying pylon API.

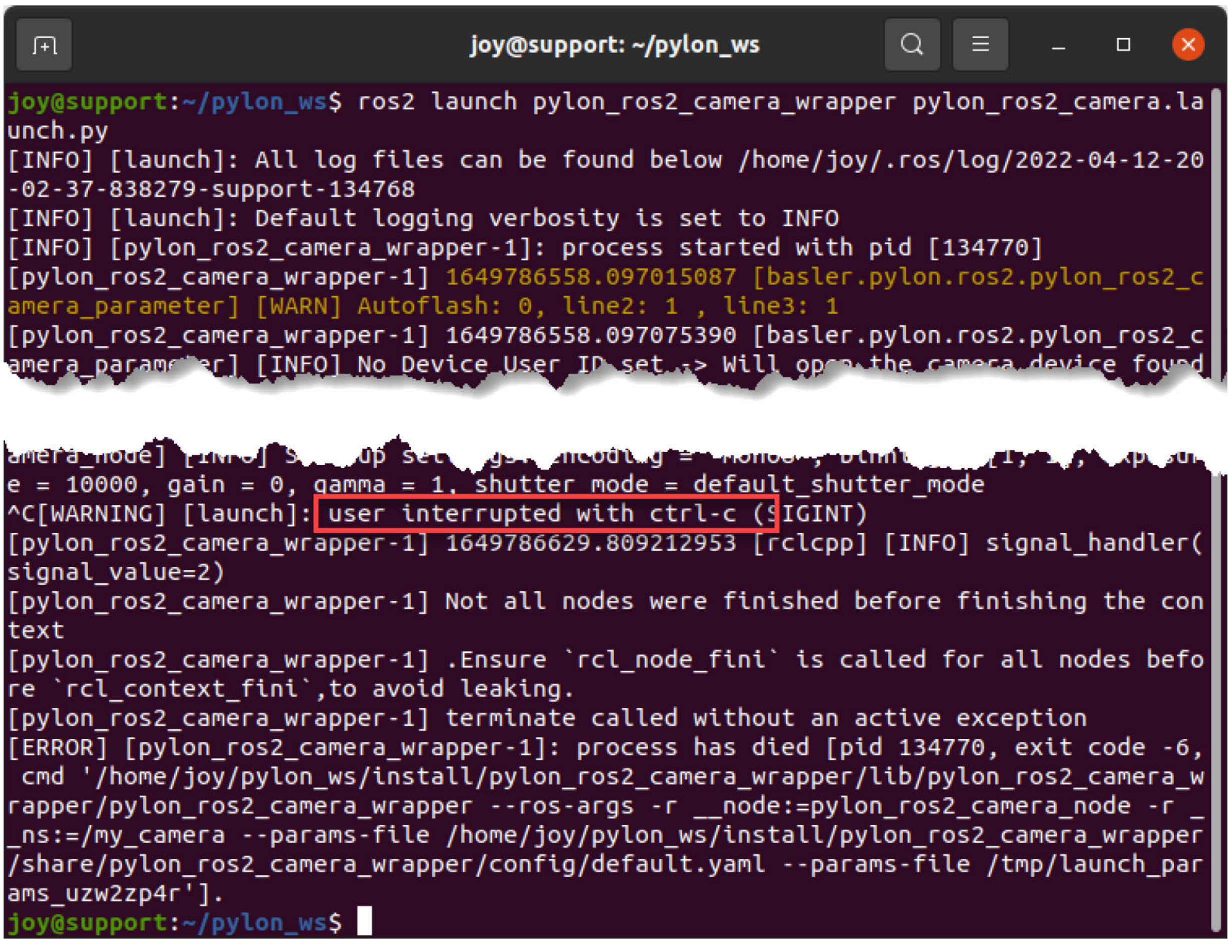
If no camera can be found, it will create an error.

If the built installation is launched, `$ ros2 launch pylon_ros2_camera_wrapper pylon_ros2_camera.launch.py`, with defaults, it will automatically use the first camera model that is found by underlaying pylon API. If no camera can be found it will create an error.



```
joy@support:~/pylon_ws$ ros2 launch pylon_ros2_camera_wrapper pylon_ros2_camera.la
unch.py
[INFO] [launch]: All log files can be found below /home/joy/.ros/log/2022-04-12-19
-56-08-810516-support-134508
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [pylon_ros2_camera_wrapper-1]: process started with pid [134510]
[pylon_ros2_camera_wrapper-1] 1649786170.445784549 [basler.pylon.ros2.pylon_ros2_c
amera_parameter] [WARN] Autoflash: 0, line2: 1 , line3: 1
[pylon_ros2_camera_wrapper-1] 1649786170.445856582 [basler.pylon.ros2.pylon_ros2_c
amera_parameter] [INFO] No Device User ID set -> Will open the camera device found
 first
[pylon_ros2_camera_wrapper-1] 1649786171.081397619 [basler.pylon.ros2.pylon_ros2_c
amera] [ERROR] No camera present
```

If a camera is found it looks as in the following screenshot.
The node is operating with the camera and provides received images via the topic channel.
It can be exited with Ctrl-C.

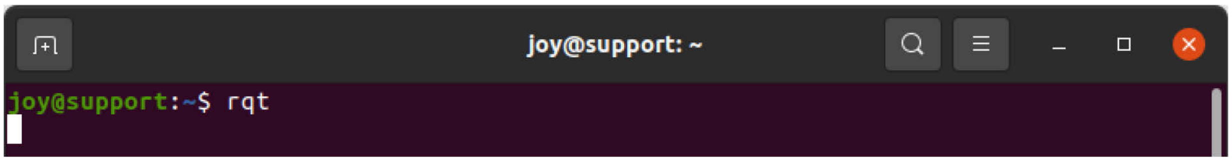


```
joy@support:~/pylon_ws$ ros2 launch pylon_ros2_camera_wrapper pylon_ros2_camera.la
unch.py
[INFO] [launch]: All log files can be found below /home/joy/.ros/log/2022-04-12-20
-02-37-838279-support-134768
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [pylon_ros2_camera_wrapper-1]: process started with pid [134770]
[pylon_ros2_camera_wrapper-1] 1649786558.097015087 [basler.pylon.ros2.pylon_ros2_c
amera_parameter] [WARN] Autoflash: 0, line2: 1 , line3: 1
[pylon_ros2_camera_wrapper-1] 1649786558.097075390 [basler.pylon.ros2.pylon_ros2_c
amera_parameter] [INFO] No Device User ID set -> Will open the camera device found

amera_node] [INFO] S[illegible]up set[illegible]encoding = [illegible][1, [illegible]
e = 10000, gain = 0, gamma = 1, shutter_mode = default_shutter_mode
^C[WARNING] [launch]: user interrupted with ctrl-c (SIGINT)
[pylon_ros2_camera_wrapper-1] 1649786629.809212953 [rclcpp] [INFO] signal_handler(
signal_value=2)
[pylon_ros2_camera_wrapper-1] Not all nodes were finished before finishing the con
text
[pylon_ros2_camera_wrapper-1] .Ensure `rcl_node_fini` is called for all nodes befo
re `rcl_context_fini`,to avoid leaking.
[pylon_ros2_camera_wrapper-1] terminate called without an active exception
[ERROR] [pylon_ros2_camera_wrapper-1]: process has died [pid 134770, exit code -6,
 cmd '/home/joy/pylon_ws/install/pylon_ros2_camera_wrapper/lib/pylon_ros2_camera_w
rapper/pylon_ros2_camera_wrapper --ros-args -r __node:=pylon_ros2_camera_node -r _
_ns:=/my_camera --params-file /home/joy/pylon_ws/install/pylon_ros2_camera_wrapper
/share/pylon_ros2_camera_wrapper/config/default.yaml --params-file /tmp/launch_par
ams_uzw2zp4r'].
joy@support:~/pylon_ws$
```

To merely view the images you can use the ROS 2 compatible version of the **image_view** node of the **image_pipeline** node stack. This node subscribes to the provided raw image topics. However, because of the more extended functionalities of image display and manipulation (see below) Basler recommends to start with the GUI -based **rqt** framework.

To open the rqt framework:

1. Open a third terminal and execute the `$ rqt` command line.



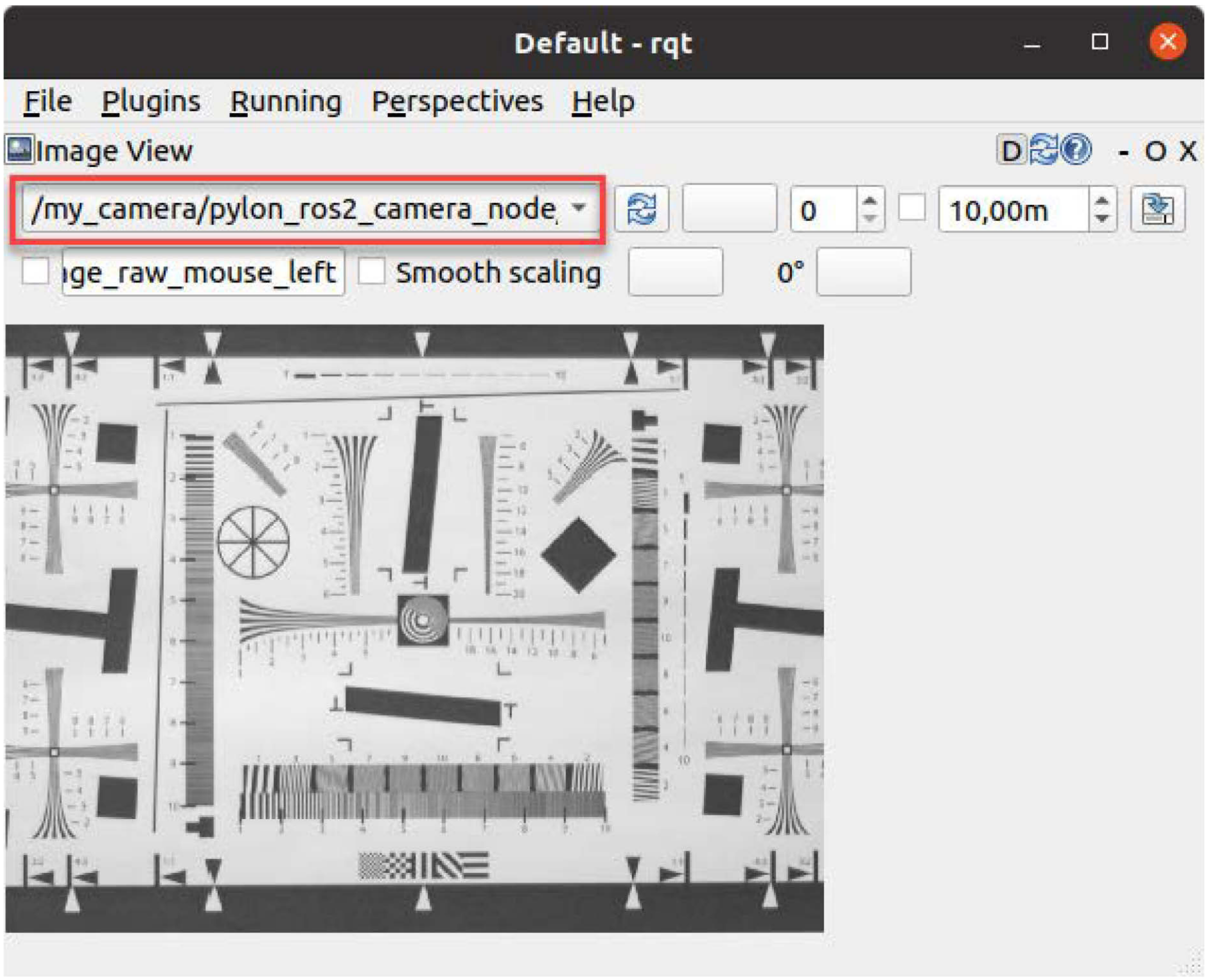
```
joy@support:~$ rqt
```

The framework GUI opens.

2. If not yet done, open the **Plugins -> Visualization** menu and select **Image View**. This enables permanent image display.

An image viewer control opens where the camera's live images can be seen, zoomed, and saved.

3. Apply the `/my_camera/pylon_ros2_camera_node/image_raw` topic as described in the screenshot below.



The camera interfacing is complete.

3 Controlling the Camera

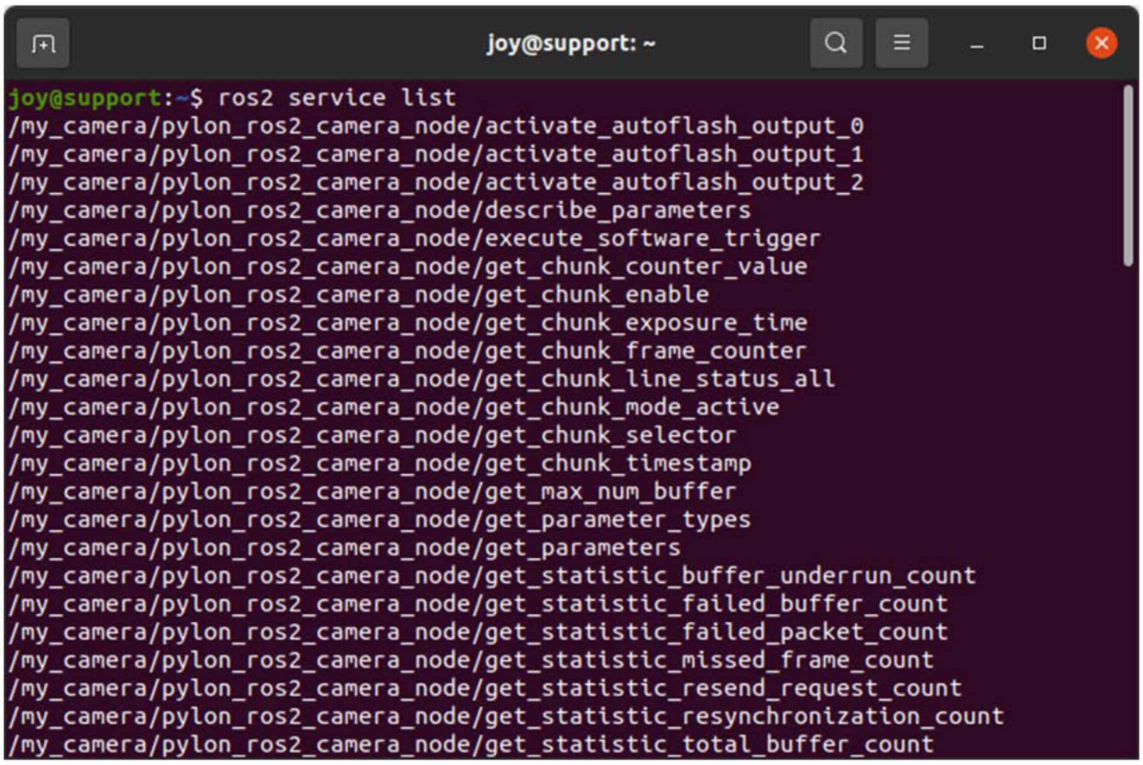
To control the cameras by setting camera parameters, so-called services are used. Contrary to single message topics that are provided after subscription, the services are able to handle request reply communication. Therefore, a pair of messages defines them. Services only provide data when they are specifically called. The service abilities of the **pylon_ros2_camera** node can be seen by issuing ROS commands like **ros2 service list**, **ros2 service type <service>**, and **ros2 interface show <interface>**. See following samples of parameter settings.

To see/display the service abilities:

1. You have the following possibilities:
   Type one of the following commands:
   - **ros2 service list**
   - **ros2 service type <service>**
   - **ros2 interface show <interface>**

See the following samples of parameter settings.

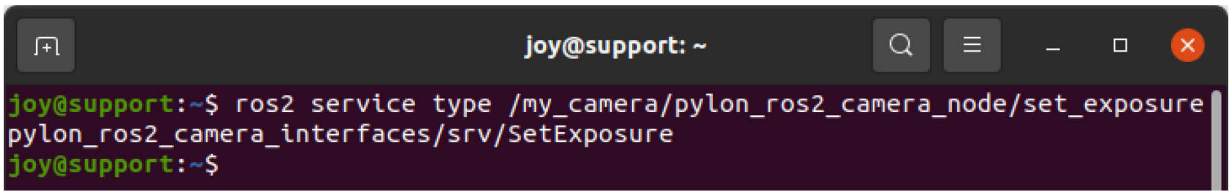
```
$ ros2 service list
```



```
joy@support:~$ ros2 service list
/my_camera/pylon_ros2_camera_node/activate_autoflash_output_0
/my_camera/pylon_ros2_camera_node/activate_autoflash_output_1
/my_camera/pylon_ros2_camera_node/activate_autoflash_output_2
/my_camera/pylon_ros2_camera_node/describe_parameters
/my_camera/pylon_ros2_camera_node/execute_software_trigger
/my_camera/pylon_ros2_camera_node/get_chunk_counter_value
/my_camera/pylon_ros2_camera_node/get_chunk_enable
/my_camera/pylon_ros2_camera_node/get_chunk_exposure_time
/my_camera/pylon_ros2_camera_node/get_chunk_frame_counter
/my_camera/pylon_ros2_camera_node/get_chunk_line_status_all
/my_camera/pylon_ros2_camera_node/get_chunk_mode_active
/my_camera/pylon_ros2_camera_node/get_chunk_selector
/my_camera/pylon_ros2_camera_node/get_chunk_timestamp
/my_camera/pylon_ros2_camera_node/get_max_num_buffer
/my_camera/pylon_ros2_camera_node/get_parameter_types
/my_camera/pylon_ros2_camera_node/get_parameters
/my_camera/pylon_ros2_camera_node/get_statistic_buffer_underrun_count
/my_camera/pylon_ros2_camera_node/get_statistic_failed_buffer_count
/my_camera/pylon_ros2_camera_node/get_statistic_failed_packet_count
/my_camera/pylon_ros2_camera_node/get_statistic_missed_frame_count
/my_camera/pylon_ros2_camera_node/get_statistic_resend_request_count
/my_camera/pylon_ros2_camera_node/get_statistic_resynchronization_count
/my_camera/pylon_ros2_camera_node/get_statistic_total_buffer_count
```

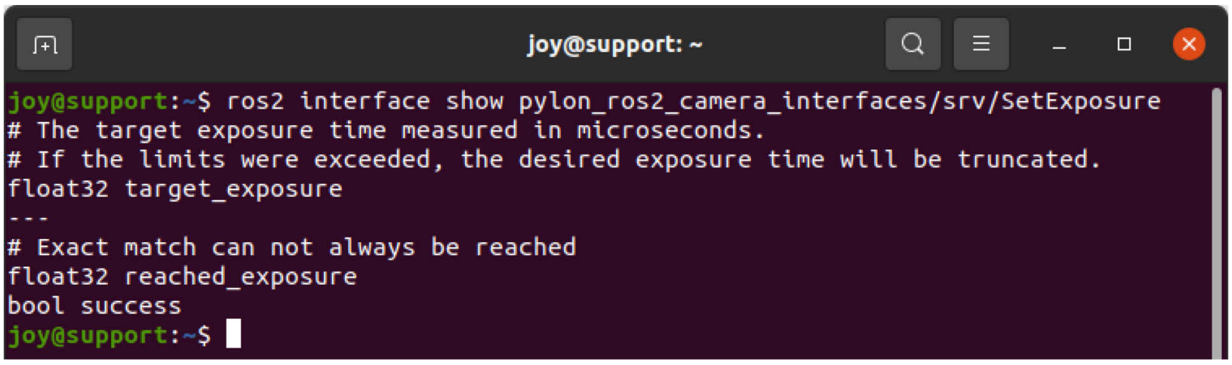
For example, the exposure time type can be:

```
$ ros2 service type /my_camera/pylon_ros2_camera_node/set_exposure
```



```
joy@support:~$ ros2 service type /my_camera/pylon_ros2_camera_node/set_exposure
pylon_ros2_camera_interfaces/srv/SetExposure
joy@support:~$
```

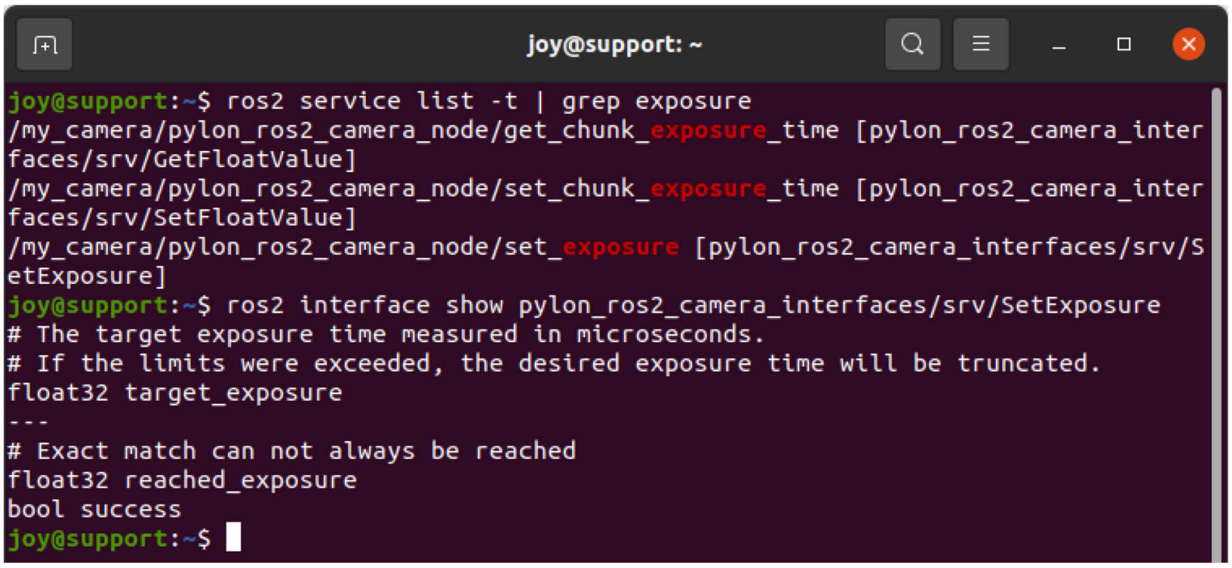
```
$ ros2 interface show pylon_ros2_camera_interfaces/srv/SetExposure
```



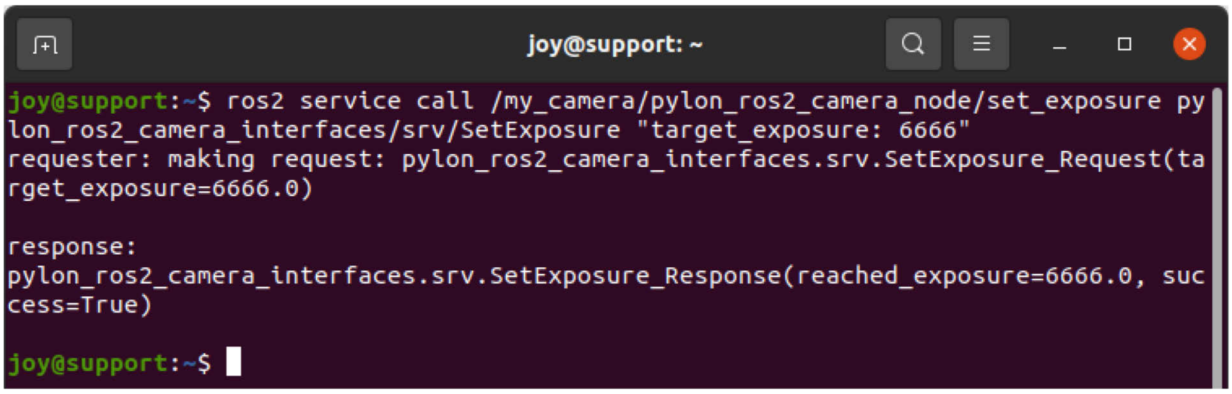
```
joy@support:~$ ros2 interface show pylon_ros2_camera_interfaces/srv/SetExposure
# The target exposure time measured in microseconds.
# If the limits were exceeded, the desired exposure time will be truncated.
float32 target_exposure
---
# Exact match can not always be reached
float32 reached_exposure
bool success
joy@support:~$
```

To find a service, the instructions **ros2 service list** and **ros2 service type** can be combined in **ros2 service list –t** with grep filter function. The execution is realized by **ros2 service call <service> <interface> “<argument(s)>”**.

```
$ ros2 service call /my_camera/pylon_ros2_camera_node/set_exposure
pylon_ros2_camera_interfaces/srv/SetExposure “target_exposure: 6666”
```

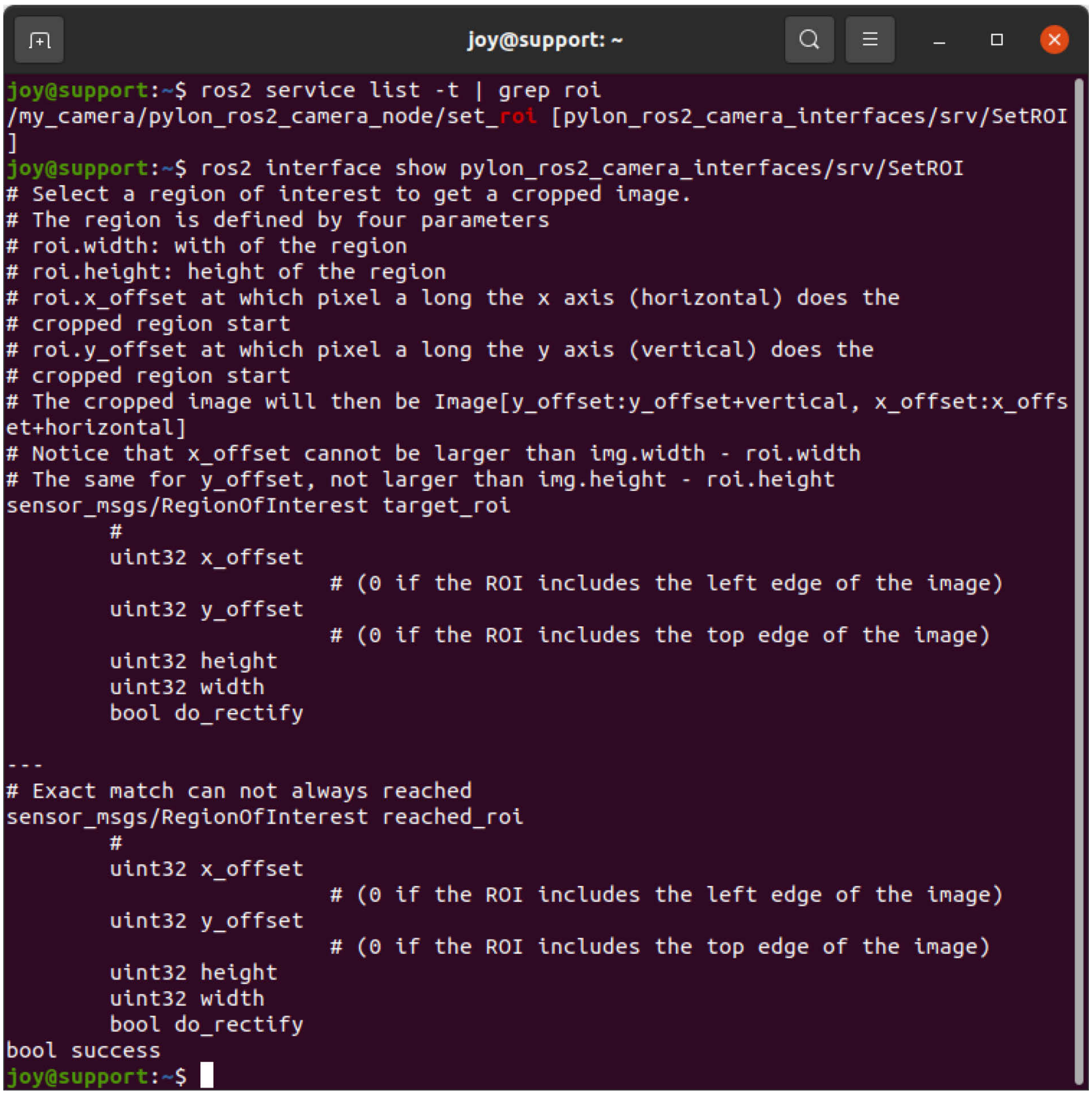


```
joy@support:~$ ros2 service list -t | grep exposure
/my_camera/pylon_ros2_camera_node/get_chunk_exposure_time [pylon_ros2_camera_inter
faces/srv/GetFloatValue]
/my_camera/pylon_ros2_camera_node/set_chunk_exposure_time [pylon_ros2_camera_inter
faces/srv/SetFloatValue]
/my_camera/pylon_ros2_camera_node/set_exposure [pylon_ros2_camera_interfaces/srv/S
etExposure]
joy@support:~$ ros2 interface show pylon_ros2_camera_interfaces/srv/SetExposure
# The target exposure time measured in microseconds.
# If the limits were exceeded, the desired exposure time will be truncated.
float32 target_exposure
---
# Exact match can not always be reached
float32 reached_exposure
bool success
joy@support:~$
```

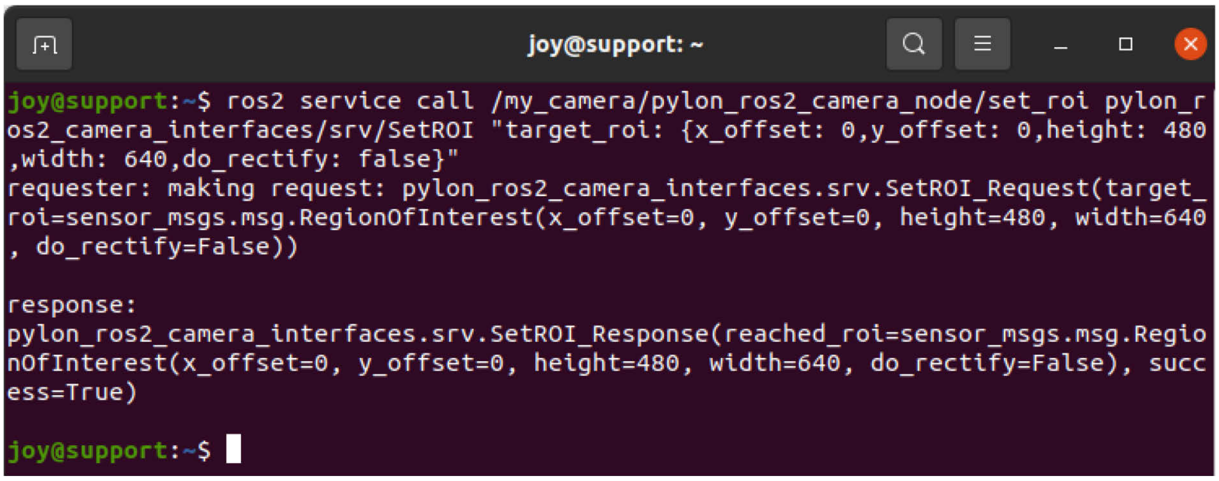


```
joy@support:~$ ros2 service call /my_camera/pylon_ros2_camera_node/set_exposure py
lon_ros2_camera_interfaces/srv/SetExposure "target_exposure: 6666"
requester: making request: pylon_ros2_camera_interfaces.srv.SetExposure_Request(ta
rget_exposure=6666.0)

response:
pylon_ros2_camera_interfaces.srv.SetExposure_Response(reached_exposure=6666.0, suc
cess=True)

joy@support:~$
```

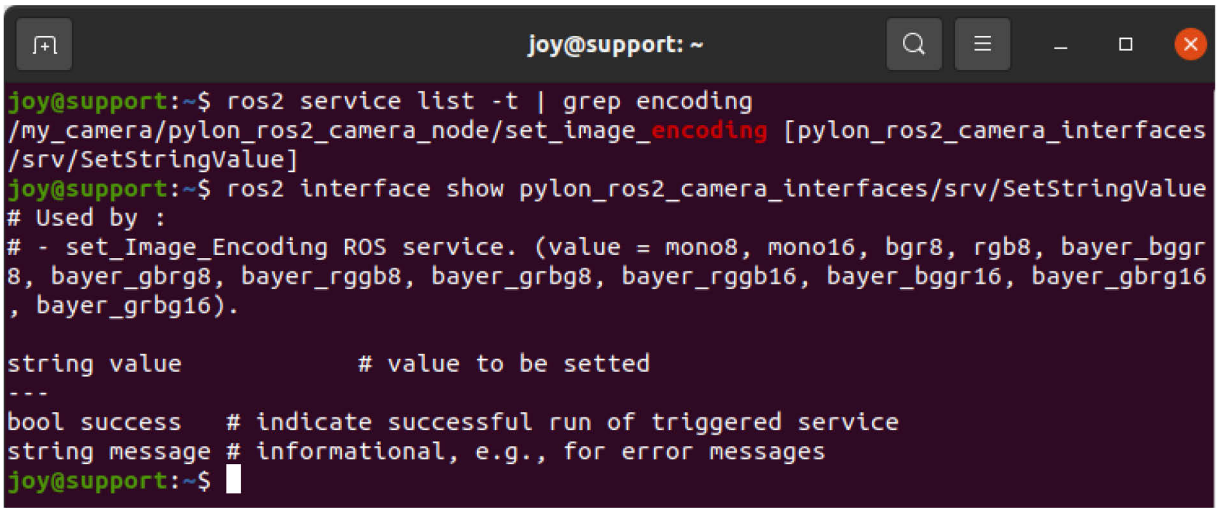
Some specific service calls concern, e.g., the definition of a ROI setup and the selection of a pixel format. See the related sample code:



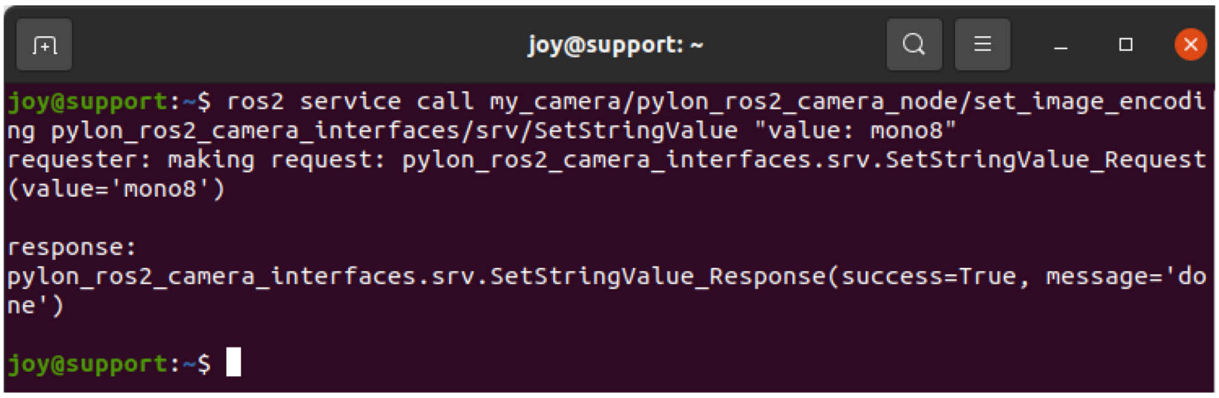
```
joy@support:~$ ros2 service list -t | grep roi
/my_camera/pylon_ros2_camera_node/set_roi [pylon_ros2_camera_interfaces/srv/SetROI
]
joy@support:~$ ros2 interface show pylon_ros2_camera_interfaces/srv/SetROI
# Select a region of interest to get a cropped image.
# The region is defined by four parameters
# roi.width: with of the region
# roi.height: height of the region
# roi.x_offset at which pixel a long the x axis (horizontal) does the
# cropped region start
# roi.y_offset at which pixel a long the y axis (vertical) does the
# cropped region start
# The cropped image will then be Image[y_offset:y_offset+vertical, x_offset:x_offs
et+horizontal]
# Notice that x_offset cannot be larger than img.width - roi.width
# The same for y_offset, not larger than img.height - roi.height
sensor_msgs/RegionOfInterest target_roi
        #
        uint32 x_offset
                                 # (0 if the ROI includes the left edge of the image)
        uint32 y_offset
                                 # (0 if the ROI includes the top edge of the image)
        uint32 height
        uint32 width
        bool do_rectify

---
# Exact match can not always reached
sensor_msgs/RegionOfInterest reached_roi
        #
        uint32 x_offset
                                 # (0 if the ROI includes the left edge of the image)
        uint32 y_offset
                                 # (0 if the ROI includes the top edge of the image)
        uint32 height
        uint32 width
        bool do_rectify
bool success
joy@support:~$
```



```
joy@support:~$ ros2 service call /my_camera/pylon_ros2_camera_node/set_roi pylon_r
os2_camera_interfaces/srv/SetROI "target_roi: {x_offset: 0,y_offset: 0,height: 480
,width: 640,do_rectify: false}"
requester: making request: pylon_ros2_camera_interfaces.srv.SetROI_Request(target_
roi=sensor_msgs.msg.RegionOfInterest(x_offset=0, y_offset=0, height=480, width=640
, do_rectify=False))

response:
pylon_ros2_camera_interfaces.srv.SetROI_Response(reached_roi=sensor_msgs.msg.Regio
nOfInterest(x_offset=0, y_offset=0, height=480, width=640, do_rectify=False), succ
ess=True)

joy@support:~$
```



```
joy@support:~$ ros2 service list -t | grep encoding
/my_camera/pylon_ros2_camera_node/set_image_encoding [pylon_ros2_camera_interfaces
/srv/SetStringValue]
joy@support:~$ ros2 interface show pylon_ros2_camera_interfaces/srv/SetStringValue
# Used by :
# - set_Image_Encoding ROS service. (value = mono8, mono16, bgr8, rgb8, bayer_bggr
8, bayer_gbrg8, bayer_rggb8, bayer_grbg8, bayer_rggb16, bayer_bggr16, bayer_gbrg16
, bayer_grbg16).

string value            # value to be setted
---
bool success    # indicate successful run of triggered service
string message # informational, e.g., for error messages
joy@support:~$
```



```
joy@support:~$ ros2 service call my_camera/pylon_ros2_camera_node/set_image_encodi
ng pylon_ros2_camera_interfaces/srv/SetStringValue "value: mono8"
requester: making request: pylon_ros2_camera_interfaces.srv.SetStringValue_Request
(value='mono8')

response:
pylon_ros2_camera_interfaces.srv.SetStringValue_Response(success=True, message='do
ne')

joy@support:~$
```

4 Driver Adjustment

ROS packages are open-source projects. The ROS 2 driver package, presented in this document serves as an example. You can, however, program your own ROS driver package according to your needs.

To get informed about latest developments of the pylon-ros2-camera driver package, access the issue tracker on the GitHub for pylon-ROS2-camera.

Revision History

Document Number	Date	Changes
AW00172901000	15 May 2022	Initial release version of this document.