Qualcomm Intelligent Robotics Product (QIRP) SDK 2.0 User Guide

80-70018-265 AC

April 29, 2025

Contents

1 Introduction

The Qualcomm® Intelligent Robotics Product (QIRP) SDK 2.0 is a collection of components that enable you to develop robotic features on Qualcomm platforms. This SDK is applicable to the Qualcomm Linux releases.

The QIRP SDK provides the following:

- Reference code in Robot Operating System (ROS) packages to develop robotic applications.
- E2E scenario samples to evaluate robotic platforms.
- Integrated cross-compile toolchain, which includes common build tools, such as `aarch64-oe-linux-gcc`, `make`, `cmake`, and ROS core.
- Tools and scripts to speed up the development.

This document guides you through developing your first sample application. It explains how to:

- Generate the QIRP SDK
- Install the QIRP SDK
- Run sample applications

1.1 Supported platform

Hardware	Availability	Quick start
Qualcomm Dragonwing™ RB3 Gen 2 Vision Development Kit	Public	RB3 Gen 2 Development Kit Quick Start Guide
Qualcomm Dragonwing™ IQ-9075 Evaluation Kit	Authorized users	IQ-9075 Evaluation Kit Quick Start Guide
Qualcomm® IQ-8 Beta Evaluation Kit	Authorized users	IQ-8 Beta Evaluation Kit Quick Start Guide

Note:

- For information about access levels, including `public`, `registered`, and `authorized`, see Working with Qualcomm.

- For information about the latest release of QIRP SDK, including new features and release tags, see Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.

1.2 QIRP SDK workflows

Depending on your user profile, you need to follow different workflows to use the QIRP SDK. For more information about the unregistered users and registered users, see the Qualcomm® Linux Build Guide.

QIRP user profile	Account	Workflows	Access
Unregistered user	No account	• Quick start (using the prebuilt package) • **Build and install**: Build with GitHub • **Use samples** : Open-source samples • Develop a robotic application	**QIRP SDK** • Prebuilt QIRP SDK + Robotics image • QIRP SDK basic layers **Qualcomm Linux** • Qualcomm Linux basic layers
Registered user	Any email account	• Quick start (using the prebuilt package) • **Build and install**: – Build with QSC Launcher – Build with QSC-CLI – Build with GitHub • **Use samples** : Open-source samples • Develop a robotic application	**QIRP SDK** • Prebuilt QIRP SDK + Robotics image • QIRP SDK basic layers **Qualcomm Linux** • Qualcomm Linux basic layers

QIRP user profile	Account	Workflows	Access
Authorized user	Organization account with a license	• Quick start (using the prebuilt package) • **Build and install**: – Build with QSC Launcher – Build with QSC-CLI – Build with GitHub – Build with GitHub (firmware and extras) • **Use samples** : Open-source samples • Develop a robotic application	**QIRP SDK** • Prebuilt QIRP SDK + Robotics image • QIRP SDK basic and extra layers **Qualcomm Linux** • Qualcomm Linux basic and extra layers • Qualcomm Linux firmware sources

2 Quick start

This information is applicable to unregistered users and all users who want to understand the basic workflow of QIRP SDK.

2.1 Download and use the prebuilt package

Get an out-of-the-box experience with the prebuilt robotics package for

- Qualcomm Dragonwing™ RB3 Gen 2 Vision Development Kit
- Qualcomm Dragonwing™ IQ-9075 Evaluation Kit

This information provides instructions on how to download and use the robotics prebuilt package, allowing you to get an out-of-the-box experience without the compiling process. The prebuilt package includes:

- **Robotics image**: An image based on the Qualcomm Linux release with the ROS core packages added and the QIRP SDK included by default. You can directly use the robotics image to get an out-of-the-box experience.
- **QIRP SDK**: Provides not only a runtime installation package with the out-of-the-box experience, but also a cross-compilation toolchain. Using that toolchain, you can develop an application in a shorter time based on the sample code.
- **Robotics eSDK**: Provides the Yocto toolchain for building the robotics image. For details, see Build the robotics image with the prebuilt robotics eSDK.

Steps:

RB3 Gen 2 Vision Development Kit

1. Download the prebuilt package.
   - RB3 Gen 2 Vision Development Kit x86 image

```
wget https://artifacts.codelinaro.org/artifactory/qli-ci/
flashable-binaries/qirpsdk/qcs6490-rb3gen2-vision-kit/x86-
qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.zip
```

- RB3 Gen 2 Vision Development Kit Arm® image

```
wget https://artifacts.codelinaro.org/artifactory/qli-ci/
flashable-binaries/qirpsdk/qcs6490-rb3gen2-vision-kit/arm-
qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.zip
```

2. Extract the package with the following command:
   - RB3 Gen 2 Vision Development Kit x86 image

```
unzip x86-qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.
1.zip
```

   - RB3 Gen 2 Vision Development Kit Arm® image

```
unzip arm-qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.
1.zip
```

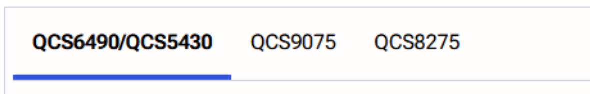
The following table lists the contents and respective locations:

Table : Contents and locations of QCS6490 prebuilt items

Content type	File path
Robotics image	`<decompressed_workspace>/target/qcs6490-rb3gen2-vision-kit/qcom-robotics-full-image`
QIRP SDK	`<decompressed_workspace>/target/qcs6490-rb3gen2-vision-kit/qirpsdk_artifacts/qcs6490-rb3gen2-vision-kit/qirp-sdk_<version>.tar.gz`
Robotics eSDK	`<decompressed_workspace>/target/qcs6490-rb3gen2-vision-kit/sdk/qcom-robotics-ros2-jazzy-x86_64-qcom-robotics-full-image-armv8-2a-qcs6490-rb3gen2-vision-kit-toolchain-ext-2.2.0.sh`

3. Build a standalone QDL by completing the following steps as described in Build a standalone QDL.
   a. Install dependent packages.
   b. Download and compile the Linux flashing tool (QDL).
4. For a new device, update the `udev` rules with the steps in Qualcomm Linux Build Guide: Update udev rules.
5. Force the device to enter EDL mode to enable software flashing with the manual steps in Qualcomm Linux Build Guide: Move to EDL mode.

Manual



6. Flash the robotics image to the device using the generated QDL.

   Note: The QDL for robotics SDK is generated in a different path. Ensure to use the following command to flash QDL to the device, where `<prebuilt_package_extracted_path>` indicates the root path of the extracted prebuilt package.

```
./qdl --storage ufs --include <prebuilt_package_extracted_path>/
target/qcs6490-rb3gen2-vision-kit/qcom-robotics-full-image \
<prebuilt_package_extracted_path>/target/qcs6490-rb3gen2-vision-
kit/qcom-robotics-full-image/prog_firehose_ddr.elf \
<prebuilt_package_extracted_path>/target/qcs6490-rb3gen2-vision-
kit/qcom-robotics-full-image/rawprogram*.xml \
<prebuilt_package_extracted_path>/target/qcs6490-rb3gen2-vision-
kit/qcom-robotics-full-image/patch*.xml
```

7. Install the prebuilt QIRP SDK to the device.

   a. On the host computer, move to the artifacts directory and decompress the package using the `tar` command.

   ```
   cd <prebuilt_package_extracted_path>/target/qcs6490-rb3gen2-
   vision-kit/qirpsdk_artifacts/qcs6490-rb3gen2-vision-kit
   tar -zxf qirp-sdk_<qirp_version>.tar.gz
   ```

   The `qirp-sdk` directory is generated.

   b. Enable `SSH` in ‘Permissive’ mode with the steps mentioned in Sign in using SSH.

   c. To deploy the QIRP artifacts, push the QIRP files to the device using the following commands:

   ```
   cd <prebuilt_package_extracted_path>/target/qcs6490-rb3gen2-
   vision-kit/qirpsdk_artifacts/qcs6490-rb3gen2-vision-kit/qirp-
   sdk
   scp ./runtime/qirp-sdk.tar.gz root@[ip-addr]:/opt/
   ssh root@[ip-addr]
   (ssh) mount -o remount,rw /usr
   (ssh) cd /opt && tar -zxf ./qirp-sdk.tar.gz
   (ssh) chmod +x /opt/scripts/*.sh
   (ssh) cd /opt/scripts && ./install.sh
   ```

IQ-9075 Evaluation Kit

1. Download the prebuilt package.

   - IQ-9075 Evaluation Kit x86 image

   ```
   wget https://artifacts.codelinaro.org/artifactory/qli-ci/flashable-binaries/qirpsdk/qcs9075-rb8-core-kit/x86-qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.zip
   ```

   - IQ-9075 Evaluation Kit Arm® image

   ```
   wget https://artifacts.codelinaro.org/artifactory/qli-ci/flashable-binaries/qirpsdk/qcs9075-rb8-core-kit/arm-qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.zip
   ```

2. Extract the package with the following command:

   - IQ-9075 Evaluation Kit x86 image

   ```
   unzip x86-qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.zip
   ```

   - IQ-9075 Evaluation Kit Arm® image

   ```
   unzip arm-qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.zip
   ```

   The following table lists the contents and respective locations:

   Table : Contents and locations of QCS9075 prebuilt items

Content type	File path
Robotics image	<decompressed_workspace>/target/qcs9075-rb8-core-kit/qcom-robotics-full-image
QIRP SDK	<decompressed_workspace>/target/qcs9075-rb8-core-kit/qirpsdk_artifacts/qcs9075-rb8-core-kit/qirp-sdk_<version>.tar.gz
Robotics eSDK	<decompressed_workspace>/target/qcs9075-rb8-core-kit/sdk/qcom-robotics-ros2-jazzy-x86_64-qcom-robotics-full-image-armv8-2a-qcs9075-rb8-core-kit-toolchain-ext-2.2.0.sh

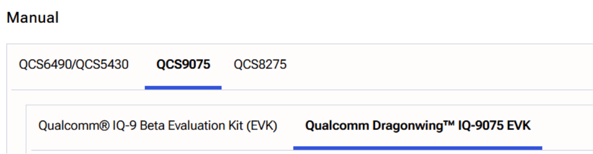
3. Build a standalone QDL by completing the following steps as described in Build a standalone QDL.

   a. Install dependent packages.

  b. Download and compile the Linux flashing tool (QDL).

4. For a new device, update the `udev` rules with the steps in Qualcomm Linux Build Guide: Update udev rules.

5. Force the device to enter EDL mode to enable software flashing with the manual steps described in Move to EDL mode.



6. Flash the robotics image to the device using the generated QDL.

Note: The QDL for robotics SDK is generated in a different path. Ensure to use the following command to flash QDL to the device, where <prebuilt_package_extracted_path> indicates the root path of the extracted prebuilt package.

```
./qdl --storage ufs --include <prebuilt_package_extracted_path>/
target/qcs9075-rb8-core-kit/qcom-robotics-full-image \
<prebuilt_package_extracted_path>/target/qcs9075-rb8-core-kit/
qcom-robotics-full-image/prog_firehose_ddr.elf \
<prebuilt_package_extracted_path>/target/qcs9075-rb8-core-kit/
qcom-robotics-full-image/rawprogram*.xml \
<prebuilt_package_extracted_path>/target/qcs9075-rb8-core-kit/
qcom-robotics-full-image/patch*.xml
```

7. Install the prebuilt QIRP SDK to the device.

  a. On the host computer, move to the artifacts directory and decompress the package using the `tar` command.

```
cd <prebuilt_package_extracted_path>/target/qcs9075-rb8-core-
kit/qirpsdk_artifacts/qcs9075-rb8-core-kit
tar -zxf qirp-sdk_<qirp_version>.tar.gz
```

  The `qirp-sdk` directory is generated.

  b. Enable `SSH` in ‘Permissive’ mode with the steps mentioned in Sign in using SSH.

  c. To deploy the QIRP artifacts, push the QIRP files to the device using the following commands:

```
cd <prebuilt_package_extracted_path>/target/qcs9075-rb8-core-
kit/qirpsdk_artifacts/qcs9075-rb8-core-kit/qirp-sdk
scp ./runtime/qirp-sdk.tar.gz root@[ip-addr]:/opt/
```

```
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) cd /opt && tar -zxf ./qirp-sdk.tar.gz
(ssh) chmod +x /opt/scripts/*.sh
(ssh) cd /opt/scripts && ./install.sh
```

Next steps

You can now do the following:

- Run robotic sample applications
- Develop a robotic application

2.2 Run robotic sample applications

The QIRP SDK provides the sample applications that you can run and refer to the sample code to write your own robotics and ROS applications.

Ensure that you run the QIRP SDK using a robotics image. Before running the sample apps, flash the prebuilt robotics image and install the QIRP SDK.

RPLIDAR ROS node

The RPLIDAR package provides a basic device handling for 2D laser scanners RPLIDAR A1, A2 and A3. RPLIDAR is a low-cost lidar sensor suitable for the indoor robotic SLAM application. This sample application uses the RPLIDAR A3 as an example.

Prerequisites:

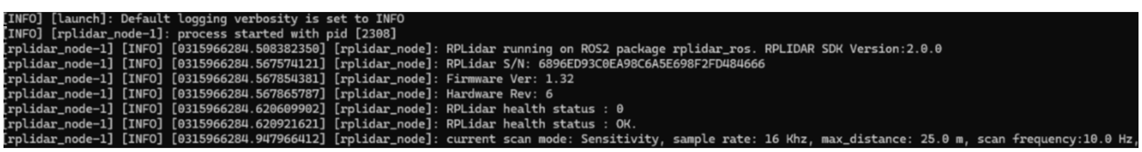
- Ensure that the RPLIDAR A3 is available and you have read the user guide of A3 for its basic information. For details about RPLIDAR A3, see the RPLIDAR A3 product page.
- You have downloaded and flashed the prebuilt robotics image. See Download and use the prebuilt package.
- You have enabled `SSH` in 'Permissive' mode with the steps mentioned in Sign in using SSH.

Steps:

1. Power on the device.
2. Connect your RPLIDAR A3M1 to the device.
3. Start a terminal and run the following commands that set up QIRP SDK and ROS2 environment on the device and run RPLIDAR.

```
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) source /usr/bin/ros_setup.bash
(ssh) ros2 launch rplidar_ros rplidar_a3_launch.py
```

The following log is shown in the terminal.



```
[INFO] [launch]: Default logging verbosity is set to INFO
[INFO] [rplidar_node-1]: process started with pid [2308]
[rplidar_node-1] [INFO] [0315966284.508382350] [rplidar_node]: RPLidar running on ROS2 package rplidar_ros. RPLIDAR SDK Version:2.0.0
[rplidar_node-1] [INFO] [0315966284.567574121] [rplidar_node]: RPLidar S/N: 6896ED93C0EA98C6A5E698F2FD484666
[rplidar_node-1] [INFO] [0315966284.567854381] [rplidar_node]: Firmware Ver: 1.32
[rplidar_node-1] [INFO] [0315966284.567865787] [rplidar_node]: Hardware Rev: 6
[rplidar_node-1] [INFO] [0315966284.620609902] [rplidar_node]: RPLidar health status : 0
[rplidar_node-1] [INFO] [0315966284.620921621] [rplidar_node]: RPLidar health status : OK.
[rplidar_node-1] [INFO] [0315966284.947966412] [rplidar_node]: current scan mode: Sensitivity, sample rate: 16 Khz, max_distance: 25.0 m, scan frequency:10.0 Hz,
```

Note: For more details about the 2D lidar ROS node, see the SLAMTEC LIDAR ROS2 Package.

Orbbec camera ROS node

The `orbbec-camera` ROS node makes the Orbbec camera 335L work properly in RGB or depth mode. It can generate the RGB and depth information by ROS topics.

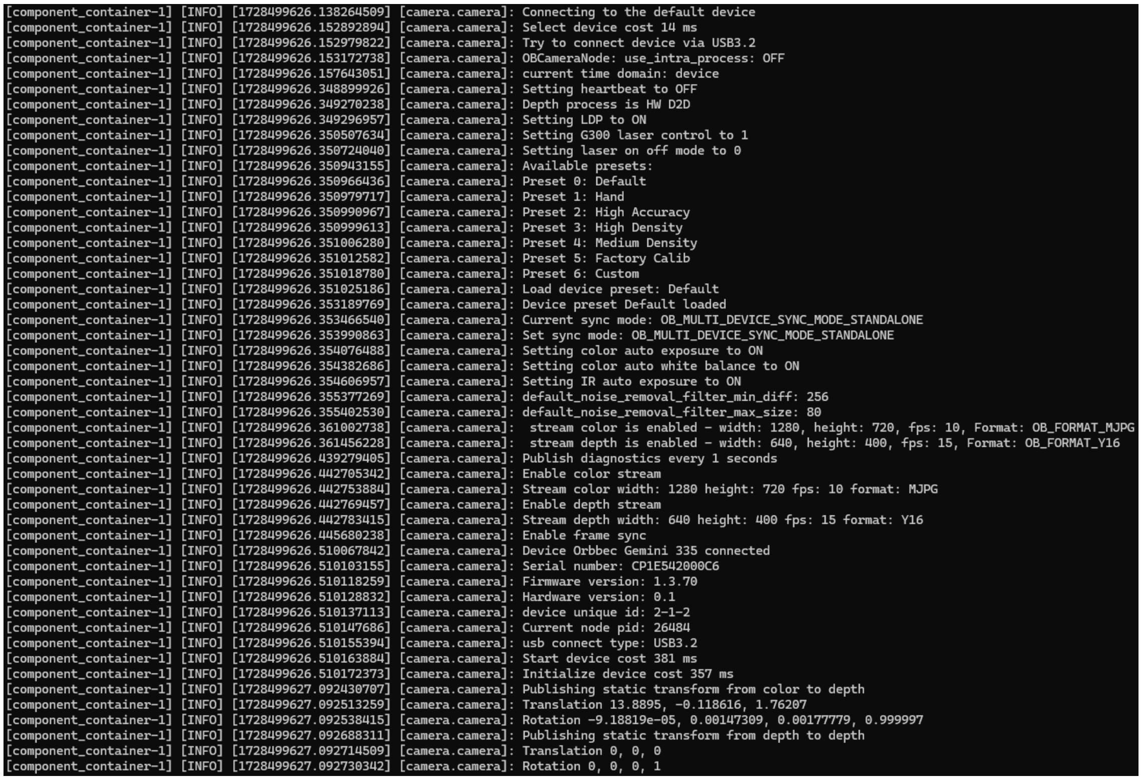
Prerequisites:

- The Orbbec camera 335L is available. For details, see Orbbec Gemini 335L product page.
- You have downloaded and flashed the prebuilt robotics image. See Download and use the prebuilt package.
- You have enabled `SSH` in ‘Permissive’ mode with the steps mentioned in Sign in using SSH.

Steps:

1. Power on the device.
2. Connect your orbbec camera 335L to the device.
3. Start a terminal and run the following commands to set up QIRP SDK and ROS2 environment and launch `orbbec-camera` ROS node.

```
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) cd /usr/share/orbbec_camera/scripts/
(ssh) bash install_udev_rules.sh
(ssh) udevadm control --reload-rules && udevadm trigger
(ssh) source ../scripts/dds_config.sh
(ssh) ros2 launch orbbec_camera gemini_330_series.launch.py
```



```
[component_container-1] [INFO] [1728499626.138264509] [camera.camera]: Connecting to the default device
[component_container-1] [INFO] [1728499626.152892894] [camera.camera]: Select device cost 14 ms
[component_container-1] [INFO] [1728499626.152979822] [camera.camera]: Try to connect device via USB3.2
[component_container-1] [INFO] [1728499626.153172738] [camera.camera]: OBCameraNode: use_intra_process: OFF
[component_container-1] [INFO] [1728499626.157643051] [camera.camera]: current time domain: device
[component_container-1] [INFO] [1728499626.348899926] [camera.camera]: Setting heartbeat to OFF
[component_container-1] [INFO] [1728499626.349270238] [camera.camera]: Depth process is HW D2D
[component_container-1] [INFO] [1728499626.349296957] [camera.camera]: Setting LDP to ON
[component_container-1] [INFO] [1728499626.350507634] [camera.camera]: Setting G300 laser control to 1
[component_container-1] [INFO] [1728499626.350724040] [camera.camera]: Setting laser on off mode to 0
[component_container-1] [INFO] [1728499626.350943155] [camera.camera]: Available presets:
[component_container-1] [INFO] [1728499626.350966436] [camera.camera]: Preset 0: Default
[component_container-1] [INFO] [1728499626.350979717] [camera.camera]: Preset 1: Hand
[component_container-1] [INFO] [1728499626.350990967] [camera.camera]: Preset 2: High Accuracy
[component_container-1] [INFO] [1728499626.350999613] [camera.camera]: Preset 3: High Density
[component_container-1] [INFO] [1728499626.351006280] [camera.camera]: Preset 4: Medium Density
[component_container-1] [INFO] [1728499626.351012582] [camera.camera]: Preset 5: Factory Calib
[component_container-1] [INFO] [1728499626.351018780] [camera.camera]: Preset 6: Custom
[component_container-1] [INFO] [1728499626.351025186] [camera.camera]: Load device preset: Default
[component_container-1] [INFO] [1728499626.353189769] [camera.camera]: Device preset Default loaded
[component_container-1] [INFO] [1728499626.353466540] [camera.camera]: Current sync mode: OB_MULTI_DEVICE_SYNC_MODE_STANDALONE
[component_container-1] [INFO] [1728499626.353990863] [camera.camera]: Set sync mode: OB_MULTI_DEVICE_SYNC_MODE_STANDALONE
[component_container-1] [INFO] [1728499626.354076488] [camera.camera]: Setting color auto exposure to ON
[component_container-1] [INFO] [1728499626.354382686] [camera.camera]: Setting color auto white balance to ON
[component_container-1] [INFO] [1728499626.354606957] [camera.camera]: Setting IR auto exposure to ON
[component_container-1] [INFO] [1728499626.355377269] [camera.camera]: default_noise_removal_filter_min_diff: 256
[component_container-1] [INFO] [1728499626.355402530] [camera.camera]: default_noise_removal_filter_max_size: 80
[component_container-1] [INFO] [1728499626.361002738] [camera.camera]:  stream color is enabled - width: 1280, height: 720, fps: 10, Format: OB_FORMAT_MJPG
[component_container-1] [INFO] [1728499626.361456228] [camera.camera]:  stream depth is enabled - width: 640, height: 400, fps: 15, Format: OB_FORMAT_Y16
[component_container-1] [INFO] [1728499626.439279405] [camera.camera]: Publish diagnostics every 1 seconds
[component_container-1] [INFO] [1728499626.442705342] [camera.camera]: Enable color stream
[component_container-1] [INFO] [1728499626.442753884] [camera.camera]: Stream color width: 1280 height: 720 fps: 10 format: MJPG
[component_container-1] [INFO] [1728499626.442769457] [camera.camera]: Enable depth stream
[component_container-1] [INFO] [1728499626.442783415] [camera.camera]: Stream depth width: 640 height: 400 fps: 15 format: Y16
[component_container-1] [INFO] [1728499626.445680238] [camera.camera]: Enable frame sync
[component_container-1] [INFO] [1728499626.510067842] [camera.camera]: Device Orbbec Gemini 335 connected
[component_container-1] [INFO] [1728499626.510103155] [camera.camera]: Serial number: CP1E542000C6
[component_container-1] [INFO] [1728499626.510118259] [camera.camera]: Firmware version: 1.3.70
[component_container-1] [INFO] [1728499626.510128832] [camera.camera]: Hardware version: 0.1
[component_container-1] [INFO] [1728499626.510137113] [camera.camera]: device unique id: 2-1-2
[component_container-1] [INFO] [1728499626.510147686] [camera.camera]: Current node pid: 26484
[component_container-1] [INFO] [1728499626.510155394] [camera.camera]: usb connect type: USB3.2
[component_container-1] [INFO] [1728499626.510163884] [camera.camera]: Start device cost 381 ms
[component_container-1] [INFO] [1728499626.510172373] [camera.camera]: Initialize device cost 357 ms
[component_container-1] [INFO] [1728499627.092430707] [camera.camera]: Publishing static transform from color to depth
[component_container-1] [INFO] [1728499627.092513259] [camera.camera]: Translation 13.8895, -0.118616, 1.76207
[component_container-1] [INFO] [1728499627.092538415] [camera.camera]: Rotation -9.18819e-05, 0.00147309, 0.00177779, 0.999997
[component_container-1] [INFO] [1728499627.092688311] [camera.camera]: Publishing static transform from depth to depth
[component_container-1] [INFO] [1728499627.092714509] [camera.camera]: Translation 0, 0, 0
[component_container-1] [INFO] [1728499627.092730342] [camera.camera]: Rotation 0, 0, 0, 1
```

Note: For more details about `orbbec-camera`, see OrbbecSDK ROS2 Wrapper GitHub repo.

IMU ROS node

The Qualcomm Sensor See Framework provides the IMU data obtained from the IMU driver using DSP. The `qrb_ros_imu` uses this framework to get the latest IMU data with less latency.

Prerequisites:

- You have downloaded and flashed the prebuilt robotics image. See Download and use the prebuilt package.
- You have downloaded and decompressed the prebuilt QIRP SDK.
- You have enabled `SSH` in 'Permissive' mode with the steps mentioned in Sign in using SSH.

Steps:

1. Set up the cross-compile environment.

```
cd <qirp_decompressed_workspace>/qirp-sdk
source setup.sh
```

2. Build the sample.

```
cd qirp-samples/demos/platform/qrb_ros_imu
colcon build --continue-on-error --cmake-args \
   -DCMAKE_TOOLCHAIN_FILE=${OE_CMAKE_TOOLCHAIN_FILE} \
   -DPYTHON_EXECUTABLE=${OECORE_NATIVE_SYSROOT}/usr/bin/python3
\
   -DPython3_NumPy_INCLUDE_DIR=${OECORE_NATIVE_SYSROOT}/usr/lib/
python3.12/site-packages/numpy/core/include \
   -DCMAKE_MAKE_PROGRAM=/usr/bin/make \
   -DBUILD_TESTING=OFF
```

3. Install the IMU to the device.

```
cd qirp-samples/demos/platform/qrb_ros_imu/install/qrb_ros_imu
tar czvf qrb_ros_imu.tar.gz include lib share
scp qrb_ros_imu.tar.gz root@[ip-addr]:/opt/
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) tar --no-same-owner -zxf /opt/qrb_ros_imu.tar.gz -C /usr/
```

4. In terminal 1, run the IMU node.

   Value range of `ROS_DOMAIN_ID`: [0, 232]

```
(ssh) mount -o remount,rw /usr
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) export ROS_DOMAIN_ID=xx
(ssh) source /usr/bin/ros_setup.bash
(ssh) ros2 run qrb_ros_imu imu_node
```

```
sh-5.1# ros2 run qrb_ros_imu imu_node
sensor client recv from msg. User set sample_rate: 200 adjusted sample_rate: 200 len: 8
[INFO] [0315964870.050204973] [imu_node]: imu client connect success
[INFO] [0315964870.052510910] [imu_node]: imu component running...
```

5. In terminal 2, verify the ROS topic and message.

   - Verify ROS topic.

```
(ssh) mount -o remount,rw /usr
(ssh) source /usr/share/qirp-setup.sh
(ssh) export ROS_DOMAIN_ID=xx
(ssh) source /usr/bin/ros_setup.bash
(ssh) ros2 topic list
```

   Results:

```
/imu
 /parameter_events
 /rosout
```

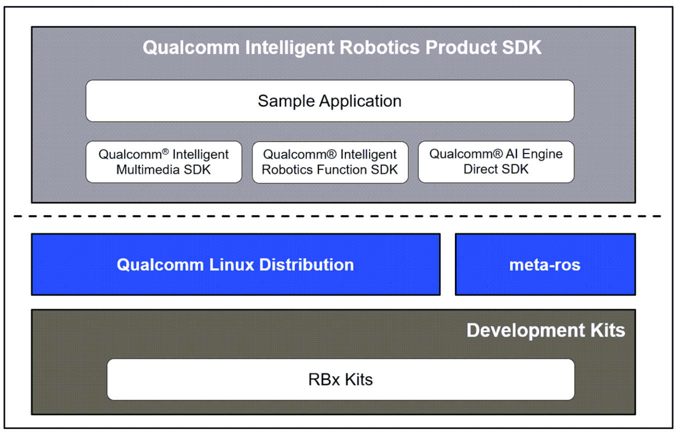
- Verify ROS message.

```
(ssh) ros2 topic list
(ssh) ros2 topic echo /imu
```

```
sh-5.1# ros2 topic echo /imu
header:
  stamp:
    sec: 315964926
    nanosec: 77376119
  frame_id: imu
orientation:
  x: 0.0
  y: 0.0
  z: 0.0
  w: 1.0
orientation_covariance:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
angular_velocity:
  x: -0.002929474925622344
  y: -0.0035952648613601923
  z: -0.011717899702489376
angular_velocity_covariance:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
linear_acceleration:
  x: -0.12928688526153564
  y: 0.04908113554120064
  z: 9.391256332397461
linear_acceleration_covariance:
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
- 0.0
---
```

3 QIRP software architecture

The figure shows the overall software architecture of the QIRP SDK.



The QIRP SDK integrates specialized components called function SDKs, which are listed as follows:

Function SDK	**Description**
Qualcomm® Intelligent Multimedia SDK (IM SDK)	Provides Qualcomm hardware accelerated plugins for optimized application development.
Qualcomm® Intelligent Robotics Function SDK	Provides robotics functional ROS nodes for Qualcomm robotics platforms. The SDK is based on the ROS, which is an open-source, meta-operating system.
Qualcomm® AI Engine direct SDK	Provides a software architecture for AI/ML use cases on Qualcomm chipsets and AI acceleration cores. It provides a unified API with modular and extensible per-accelerator libraries, which form a reusable basis for full-stack AI solutions. It supports runtimes such as Qualcomm® Neural Processing SDK and TensorFlow Lite AI Engine Direct Delegate.

3.1 Component layers of the QIRP SDK

The QIRP SDK uses the same method as the Yocto community to arrange the source code into the following layers.

- QIRP layers: Including both robotics and Qualcomm Linux layers, open to all users.
- Extra layers: Available to authorized users with verified organization accounts, who have access to specified Qualcomm Linux firmware components and extra layers and the robotics extra layer.

Note: For more about the public, registered and authorized access levels, see the Working with Qualcomm.

QIRP layers

Table : Robotics layers

Layer	Category	Description
meta-ros	Robotics layer	A series of OpenEmbedded layers designed to add support for the Robot Operating System (ROS) for embedded Linux releases from the Yocto project.
`meta-qcom-robotics-distro`	Robotics layer	Contains the configuration information needed to generate the ROS image, including but not limited to the package groups and image recipes.
`meta-qcom-robotics`	Robotics layer	Includes the recipes for all the robotics functions.
`meta-qcom-robotics-sdk`	Robotics layer	Contains the generation/pick-up mechanism of the Qualcomm Intelligent Robotics Product (QIRP) SDK.

Note: For details about the robotics layers, see Robotics layers specifications.

Table : Qualcomm Linux layers

Layer	Category	Description
`meta-qcom`	Qualcomm Linux layer	Contains Qualcomm hardware support metadata with upstream OSS software components.
`meta-qcom-hwe`	Qualcomm Linux layer	Contains Qualcomm hardware support metadata with Qualcomm value-added software components.
`meta-qcom-distro`	Qualcomm Linux layer	Contains the distro configuration needed to generate the base image, including but not limited to the package groups and image recipes.

Layer	Category	Description
`meta-qcom-qim-product-sdk`	Dependent SDK	Provides Qualcomm's multimedia and AI SDKs based on the GStreamer framework.

Extra layers

`meta-qcom-robotics-extras`	Includes proprietary robotics functional recipes, which allow building with source.
`meta-qcom-extras`	Enables source compilation of selective Qualcomm Linux components and includes a few component binaries.

3.2 Samples list

Open-source samples

Sample application	Description
Rplidar-ros2	Provides basic device handling for 2D laser scanner RPLIDAR A1/A2/A3/S1/S2/S3.
Qrb-ros-imu	Creates the `/imu` topic to publish the inertial measurement unit (IMU) data.
Qrb-ros-system-monitor	Contains various ROS nodes to publish system status information, such as CPU loading, memory usage, and disk space.
Ocr-service	Enables a service that provides the Optical Character Recognition (OCR) function. It captures the image topic from the ROS system and publishes the result with the `ocr_topic`.
Orbbec-camera	Enables the Orbbec Gemini camera 335L to work properly in RGB or depth mode.
Qrb-ros-color-space-convert	Converts the color space between NV12 and RBG888.

4 Build and install

This section explains how to download, compile, and install the QIRP SDK.

Note: For detailed information about the workflows that various users can access, see QIRP SDK workflows.

4.1 Set up the host environment

Prepare your host computer for the build and install operations.

Prerequisites:

Ensure that the Ubuntu 22.04 host computer conforms to the Host machine requirements, with an extra 50 GB disk space required for the robotics SDK.

Steps:

1. Complete the following tasks as described in Ubuntu host setup:
   a. Install dependent packages.
   b. Set up the locales.
   c. Update the git configuration.
2. Change the `/bin/sh` symlink to point to `bash` by default.

```
sudo ln -sf /bin/bash /bin/sh
```

3. Download the Repo tool with the steps of *Install the repo utility* on the Qualcomm manifest page.
4. Install the libgtest-dev package.

```
sudo apt install libgtest-dev
```

Note: For the virtual machine (VM) build support, see Qualcomm Linux Virtual Machine Setup

Guide.

4.2 Build the software

Sync and build the software using Qualcomm Software Center (QSC) Launcher GUI, QSC command-line interface (QSC-CLI), and the GitHub workflow with standalone commands or Dockerfile.

Before you begin, register with Qualcomm.com to sign up and get access to the proprietary software required for boards supported by Qualcomm® Linux.

Note:

- For details about the three methods, see the Qualcomm® Linux Build Guide.
- For details about how to register with Qualcomm, see Working with Qualcomm.

Build the QIRP SDK and robotics image

You can use one of the following three approaches to sync and build the QIRP SDK along with the robotics image:

- Qualcomm Software Center (QSC) Launcher GUI (Build with QSC Launcher)

  Use the Qualcomm Software Center (QSC) Launcher to download, compile, and flash the QIRP SDK.

- QSC command-line interface (QSC-CLI) (Build with QSC-CLI)

  Use the Qualcomm Software Center command-line interface (QSC-CLI) to download and compile the QIRP SDK and the robotics image, and flash images using tools or commands.

- GitHub workflow with standalone commands or Dockerfile (Build with GitHub)

  Use the detailed instructions to sync and build the Qualcomm Yocto and QIRP SDK layers using standalone commands or Dockerfile

- GitHub workflow using firmware and extras (Build with GitHub (firmware and extras))

Build the robotics image only

You can also use the eSDK to build the robotics image (Build the robotics image with the prebuilt robotics eSDK).

Build with QSC Launcher

Use the Qualcomm Software Center (QSC) Launcher to download, compile, and flash the QIRP SDK.

Prerequisites:

- You have installed the QSC Launcher, with the steps in Install QSC using a GUI of *Qualcomm Linux Build Guide*.

Note: QSC Launcher uses Docker. Install Docker on the host computer if you haven't. If you are using WSL, QSC Launcher is not supported, please switch to QSC CLI.

Download and compile

To use the QSC Launcher to download and compile the QIRP SDK, follow these steps:

Steps:

1. Open and sign in the QSC Launcher desktop application according to Use QSC Launcher of *Qualcomm Linux Build Guide*.
2. Follow the steps in Use QSC Launcher to specify environment, and on the **Select Resources** page, do the following:
   a. In the *Base Workspace Path* text box, specify a directory path where you want to download the software.
   b. Select the *Software Product*.
   c. Select the *Distribution* and the *Release Tag*. The following tables list the mapping between available software products and release tags, and the mapping between distributions and access levels.

Software product and release tag

Software product	Release tag	Hardware
QCM6490.LE.1.0	See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes. For example, `r00349.1`.	Qualcomm Dragonwing™ RB3 Gen 2 Vision Development Kit
QCS9100.LE.1.0	See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes. For example, `r00214.2`.	Qualcomm Dragonwing™ IQ-9075 Evaluation Kit
QCS8300.LE.1.0	See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes. For example, `r00110.1`.	Qualcomm® IQ-8 Beta Evaluation Kit

Mapping between distributions and access levels

Access Level	Distribution	Yocto Layers
Registered developer with any email address (binaries only, without modem and GPS)	Qualcomm_ Linux.SPF.1.0|TEST| DEVICE|PB_QIRPSDK	`meta-qcom` `meta-qcom-hwe` `meta-qcom-distro` `meta-ros` `meta-qcom-robotics` `meta-qcom-robotics-distro` `meta-qcom-robotics-sdk` `meta-qcom-qim-product-sdk`

Access Level	Distribution	Yocto Layers
Authorized developer from a verified organization (binaries and selected source for firmware images, without modem and GPS)	Qualcomm_ Linux.SPF.1.0|AP| Standard|OEM|NM_ QIRPSDK	`meta-qcom` `meta-qcom-hwe` `meta-qcom-distro` `meta-qcom-extras` `meta-qcom-robotics-extras` `meta-ros` `meta-qcom-robotics` `meta-qcom-robotics-distro` `meta-qcom-robotics-sdk` `meta-qcom-qim-product-sdk`

3. Follow the subsequent steps in Use QSC Launcher to download and compile the QIRP SDK.

Outputs

Note: <Workspace_Path> is the path that you select on the **Select Resources** page.

RB3 Gen 2 Vision Development Kit

- Robotics images: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs6490-custom/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-robotics-full-image/*
- QIRP SDK: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs6490-custom/tmp-glibc/deploy/qirpsdk_artifacts/qcs6490-rb3gen2-vision-kit/qirp-sdk_<version>.tar.gz

IQ-9075 Evaluation Kit

- Robotics images: `<Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs9075-rb8-core-kit-custom/tmp-glibc/deploy/images/qcs9075-rb8-core-kit/qcom-robotics-full-image/*`
- QIRP SDK: `<Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs9075-rb8-core-kit-custom/tmp-glibc/deploy/qirpsdk_artifacts/qcs9075-rb8-core-kit/qirp-sdk_<version>.tar.gz`

IQ-8 Beta Evaluation Kit

- Robotics images: `<Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs8300-custom/tmp-glibc/deploy/images/qcs8300-ride-sx/qcom-robotics-full-image/*`
- QIRP SDK: `<Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs8300-custom/tmp-glibc/deploy/qirpsdk_artifacts/qcs8300-ride-sx/qirp-sdk_<version>.tar.gz`

Flash the QIRP SDK

To flash the robotics image to the device, follow the steps in Build and flash default configuration.

Build with QSC-CLI

Use the Qualcomm® Software Center command-line interface (QSC-CLI) to download and compile the QIRP SDK and the robotics image, and flash images using tools or commands.

Prerequisites:

- You have installed the QSC CLI, with the commands in Install QSC-CLI of *Qualcomm Linux Build Guide*.

Note: QSC CLI uses Docker. Install Docker on the host computer if you haven't.

Download

Steps:

1. Sign in `qpm-cli`.

   a. Sign in with this command.

   ```
   qpm-cli --login
   ```

   b. Verify if the `qpm-cli` login is successful.

   ```
   qpm-cli --product-list
   ```

2. Download a particular software release of the QIRP SDK with the following command. To identify the appropriate values for the command arguments, use the following tables.

   ```
   qsc-cli download --workspace-path '<absolute_workspace_path>' --product '<Product_ID>' --release '<Release_ID>' --distribution '<Distro>'
   ```

   Note: Both `--release` and `--build` options work individually to download a software package. If both options are provided, the `--release` parameter is used to trigger a download.

Table : `qsc-cli download` parameters

Parameter	Description	QSC-CLI command value
`--workspace-path`	Absolute/full workspace path	A custom value

`--product`	Product ID	Allowed product IDs
		`QCM6490.LE.1.0`
		`QCS9100.LE.1.0`
		`QCS8300.LE.1.0`

Parameter	Description	QSC-CLI command value
`--release`	Release ID	• QCM6490.LE.1.0 See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes. For example, `r00349.1`. • QCS9100.LE.1.0 See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes. For example, `r00214.2`. • QCS8300.LE.1.0 See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes. For example, `r00110.1`.

`--build`	Build ID	• QCM6490.LE.1.0
		See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.
		For example:
		`QCM6490.LE.1.0-00349-STD.PROD-1`
		• QCS9100.LE.1.0
		See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.
		For example:
		`QCS9100.LE.1.0-00214-STD.PROD-2`
		• QCS8300.LE.1.0
		See Qualcomm Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.
		For example:
		`QCS8300.LE.1.0-00110-STD.PROD-1`

`--distribution`	Distribution	See Table: Distributions and access levels.

Table : Distributions and access levels

Access level	`Distro`	Yocto layers
Registered developer with any email address (binaries only, without modem and GPS)	`Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PB_QIRPSDK`	`meta-qcom` `meta-qcom-hwe` `meta-ros` `meta-qcom-robotics` `meta-qcom-robotics-distro` `meta-qcom-robotics-sdk` `meta-qcom-qim-product-sdk`
Registered developer from a verified organization (binaries and selected source for firmware images, without modem and GPS)	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIRPSDK`	`meta-qcom` `meta-qcom-hwe` `meta-qcom-extras` `meta-qcom-robotics-extras` `meta-ros` `meta-qcom-robotics` `meta-qcom-robotics-distro` `meta-qcom-robotics-sdk` `meta-qcom-qim-product-sdk`

Note: The build and flash steps follow the Qualcomm® Linux Build Guide.

Build the QIRP with QSC-CLI downloads

To build the QIRP SDK, see Build default configuration - Compile.

Outputs

Note: `<Base_Workspace_Path>` is the argument that you provide to the `--workspace-path` parameter of the download command.

RB3 Gen 2 Vision Development Kit

- Robotics images: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs6490-custom/tmp-glibc/deploy/images/qcs6490-rb3gen2-vision-kit/qcom-robotics-full-image/*
- QIRP SDK: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs6490-custom/tmp-glibc/deploy/qirpsdk_artifacts/qcs6490-rb3gen2-vision-kit/qirp-sdk_<version>.tar.gz

IQ-9075 Evaluation Kit

- Robotics images: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs9075-rb8-core-kit-custom/tmp-glibc/deploy/images/qcs9075-rb8-core-kit/qcom-robotics-full-image/*
- QIRP SDK: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs9075-rb8-core-kit-custom/tmp-glibc/deploy/qirpsdk_artifacts/qcs9075-rb8-core-kit/qirp-sdk_<version>.tar.gz

IQ-8 Beta Evaluation Kit

- Robotics images: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs8300-custom/tmp-glibc/deploy/images/qcs8300-ride-sx/qcom-robotics-full-image/*
- QIRP SDK: <Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcs8300-custom/tmp-glibc/deploy/qirpsdk_artifacts/qcs8300-ride-sx/qirp-sdk_<version>.tar.gz

Flash the QIRP to devices

Flash the robotics image to the device, and install the QIRP SDK.

- To flash the robotics image to the device, see Flash software images.
- To install the QIRP SDK on the device, see Install the QIRP SDK on the device.

Build with GitHub

Use the detailed instructions to sync and build the Qualcomm Yocto and QIRP SDK layers using standalone commands or Dockerfile.

Prerequisites:

Ensure that you have set up the host according to Set up the host environment.

For QIRP SDK, you can use the following two methods to build:

- Build with standalone commands
- Build with Dockerfile

Build with standalone commands

Use standalone commands to download and build the QIRP SDK along with the robotics image. Supports manifest or `git clone` methods.

Steps:

1. Download the Qualcomm Yocto and QIRP SDK layers using either manifest or `git clone`.

   - **Download with manifest**: download the layers for the QIRP SDK by running the following commands.

     Note: To get the latest `<robotics-release-manifest>`, see the Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.

     ```
     cd <workspace>
     repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-scarthgap -m <robotics-release-manifest>
     repo sync -c -j8
     ```

     Example:

     The following command downloads the release with manifest `qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.xml`:

     ```
     repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-scarthgap -m qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.xml
     repo sync -c -j8
     ```

- **Download with** `git clone`:

  a. Set up the host environment and sync the latest Yocto project BSP as described in the Qualcomm Repo Manifest README file.

  b. Download the layers for the QIRP SDK based on the `<workspace>` directory of the downloaded Yocto project BSP.

```
cd <workspace>
git clone https://github.com/ros/meta-ros -b
scarthgap layers/meta-ros && cd layers/meta-ros &&
git checkout
c560699e810e60a9526f4226c2c23f8d877280c8 && cd ../.
./
git clone https://github.com/quic-yocto/meta-qcom-
robotics.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_
robotics-product-sdk-1.1 layers/meta-qcom-robotics
git clone https://github.com/quic-yocto/meta-qcom-
robotics-distro.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_
robotics-product-sdk-1.1 layers/meta-qcom-robotics-
distro
git clone https://github.com/quic-yocto/meta-qcom-
robotics-sdk.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_
robotics-product-sdk-1.1 layers/meta-qcom-robotics-
sdk
git clone https://github.com/quic-yocto/meta-qcom-
qim-product-sdk -b qcom-6.6.65-QLI.1.4-Ver.1.1_qim-
product-sdk-1.1.2 layers/meta-qcom-qim-product-sdk
```

  Results: The preceding commands download the following layers:

```
meta-qcom
meta-qcom-hwe
meta-qcom-distro
meta-ros
meta-qcom-robotics
meta-qcom-robotics-distro
meta-qcom-robotics-sdk
meta-qcom-qim-product-sdk
```

2. Set up the build environment.

Note: If you are building the QIRP SDK on the Ubuntu Server VM of an Arm64 Mac, add the variable setting `SDKMACHINE=aarch64` in the following setup command.

```
MACHINE=<Machine_name> DISTRO=<Distro_name>
SDKMACHINE=aarch64 QCOM_SELECTED_BSP=<Build_override>
source setup-robotics-environment <Build_directory>
```

- If you use the **Download with manifest** method, run these commands:

```
cd <workspace>
MACHINE=<Machine_name> DISTRO=<Distro_name>  QCOM_
SELECTED_BSP=<Build_override> source setup-robotics-
environment <Build_directory>
```

- If you use the **Download with** `git clone` method, run these commands:

```
cd <workspace>
ln -s layers/meta-qcom-robotics-distro/set_bb_env.sh .
/setup-robotics-environment
ln -s layers/meta-qcom-robotics-sdk/scripts/qirp-build
./qirp-build
MACHINE=<Machine_name> DISTRO=<Distro_name>  QCOM_
SELECTED_BSP=<Build_override> source setup-robotics-
environment <Build_directory>
```

Table : Build parameters

Parameter	RB3 Gen 2 Vision Kit	IQ-9075 Evaluation Kit	IQ-8 Beta Evaluation Kit
`Machine_name`	`qcs6490-rb3gen2-vision-kit`	`qcs9075-rb8-core-kit`	`qcs8300-ride-sx`
`Distro_name`	`qcom-robotics-ros2-jazzy`	`qcom-robotics-ros2-jazzy`	`qcom-robotics-ros2-jazzy`
`Build_override`	`custom`	• `custom` • `base`	• `custom` • `base`

Parameter	RB3 Gen 2 Vision Kit	IQ-9075 Evaluation Kit	IQ-8 Beta Evaluation Kit
Build_directory	build-qcs6490-custom	• build-qcs9075-custom • build-qcs9075-base	• build-qcs8300-custom • build-qcs8300-base

Note: For Qualcomm® IQ-8 Beta Evaluation Kit and Qualcomm Dragonwing™ IQ-9075 Evaluation Kit, the build command also supports `base` and `custom` *Build_override* values. The default override is `custom` and you can override to `base` as needed. The following example sets the `base` build override for the machine `qcs9075-rb8-core-kit`:

```
MACHINE=qcs9075-rb8-core-kit DISTRO=qcom-robotics-ros2-
jazzy QCOM_SELECTED_BSP=base source setup-robotics-
environment build-qcs9075-base
```

3. Build the robotics image and QIRP SDK artifacts.

```
../qirp-build qcom-robotics-full-image
```

Results:

RB3 Gen 2 Vision Development Kit

- QIRP SDK artifacts:

```
<workspace>/build-qcs6490-custom/tmp-glibc/deploy/
qirpsdk_artifacts/qcs6490-rb3gen2-vision-kit/qirp-
sdk_<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs6490-custom/tmp-glibc/deploy/
images/qcs6490-rb3gen2-vision-kit/qcom-robotics-
full-image
```

IQ-9075 Evaluation Kit

Example using `custom` build override

- QIRP SDK artifacts:

```
<workspace>/build-qcs9075-custom/tmp-glibc/
deploy/qirpsdk_artifacts/qcs9075-rb8-core-kit/
qirp-sdk_<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs9075-custom/tmp-glibc/
deploy/images/qcs9075-rb8-core-kit/qcom-
robotics-full-image
```

Example using `base` build override

- QIRP SDK artifacts:

```
<workspace>/build-qcs9075-base/tmp-glibc/deploy/
qirpsdk_artifacts/qcs9075-rb8-core-kit/qirp-sdk_
<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs9075-base/tmp-glibc/deploy/
images/qcs9075-rb8-core-kit/qcom-robotics-full-
image
```

IQ-8 Beta Evaluation Kit

Example using `custom` build override

- QIRP SDK artifacts:

```
<workspace>/build-qcs8300-custom/tmp-glibc/
deploy/qirpsdk_artifacts/qcs8300-ride-sx/qirp-
sdk_<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs8300-custom/tmp-glibc/
deploy/images/qcs8300-ride-sx/qcom-robotics-
full-image
```

Example using `base` build override

- QIRP SDK artifacts:

```
<workspace>/build-qcs8300-base/tmp-glibc/deploy/
qirpsdk_artifacts/qcs8300-ride-sx/qirp-sdk_
<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs8300-base/tmp-glibc/deploy/
images/qcs8300-ride-sx/qcom-robotics-full-image
```

Build with Dockerfile

Build the QIRP SDK along with the robotic image using the Dockerfile, building upon the Qualcomm Linux image.

Steps:

1. Build the Qualcomm Linux image with the steps under Build with Dockerfile.
   a. Follow Ubuntu host setup and check the host computer configuration.
   b. Follow the steps in Build BSP image.
2. Run the `docker run` command.

Note: Run the following commands inside the Qualcomm Linux build location.

```
cd <workspace_path>/qcom-download-utils/qcom-6.6.65-QLI.
1.4-Ver.1.1
bash
docker run -it -v "${HOME}/.gitconfig":"/home/${USER}/.
gitconfig" -v "${HOME}/.netrc":"/home/${USER}/.netrc" -v
$(pwd):$(pwd) -w $(pwd) qcom-6.6.65-qli.1.4-ver.1.0_22.04
/bin/bash
```

3. Download the layers for the QIRP SDK based on the `<workspace>` directory.

```
git clone https://github.com/ros/meta-ros -b scarthgap
layers/meta-ros && cd layers/meta-ros && git checkout
c560699e810e60a9526f4226c2c23f8d877280c8 && cd ../../
git clone https://github.com/quic-yocto/meta-qcom-
robotics.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-
product-sdk-1.1 layers/meta-qcom-robotics
```

```
git clone https://github.com/quic-yocto/meta-qcom-
robotics-distro.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_
robotics-product-sdk-1.1 layers/meta-qcom-robotics-distro
git clone https://github.com/quic-yocto/meta-qcom-
robotics-sdk.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-
product-sdk-1.1 layers/meta-qcom-robotics-sdk
git clone https://github.com/quic-yocto/meta-qcom-qim-
product-sdk -b qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-
sdk-1.1.2 layers/meta-qcom-qim-product-sdk
```

4. Set up the build environment.

Note: If you are building the QIRP SDK on the Ubuntu Server VM of an Arm64 Mac, add the variable setting `SDKMACHINE=aarch64` in the following setup command.

```
MACHINE=<Machine_name> DISTRO=<Distro_name>
SDKMACHINE=aarch64 QCOM_SELECTED_BSP=<Build_override>
source setup-robotics-environment <Build_directory>
```

```
ln -s layers/meta-qcom-robotics-distro/set_bb_env.sh ./
setup-robotics-environment
ln -s layers/meta-qcom-robotics-sdk/scripts/qirp-build ./
qirp-build
MACHINE=<Machine_name> DISTRO=<Distro_name>  QCOM_
SELECTED_BSP=<Build_override> source setup-robotics-
environment <Build_directory>
```

Table : Build parameters

Parameter	RB3 Gen 2 Vision Kit	IQ-9075 Evaluation Kit	IQ-8 Beta Evaluation Kit
`Machine_name`	`qcs6490-rb3gen2-vision-kit`	`qcs9075-rb8-core-kit`	`qcs8300-ride-sx`
`Distro_name`	`qcom-robotics-ros2-jazzy`	`qcom-robotics-ros2-jazzy`	`qcom-robotics-ros2-jazzy`

Parameter	RB3 Gen 2 Vision Kit	IQ-9075 Evaluation Kit	IQ-8 Beta Evaluation Kit
`Build_override`	`custom`	• `custom` • `base`	• `custom` • `base`
`Build_directory`	`build-qcs6490-custom`	• `build-qcs9075-custom` • `build-qcs9075-base`	• `build-qcs8300-custom` • `build-qcs8300-base`

5. Build the robotics image and QIRP SDK artifacts.

```
../qirp-build qcom-robotics-full-image
```

Results:

RB3 Gen 2 Vision Development Kit

- QIRP SDK artifacts:

```
<workspace>/qcom-download-utils/qcom-6.6.65-QLI.1.4-
Ver.1.1/build-qcs6490-custom/tmp-glibc/deploy/qirpsdk_
artifacts/qcs6490-rb3gen2-vision-kit/qirp-sdk_
<version>.tar.gz
```

- Robotics image:

```
<workspace>/qcom-download-utils/qcom-6.6.65-QLI.1.4-
Ver.1.1/build-qcs6490-custom/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-robotics-full-image
```

IQ-9075 Evaluation Kit

Example using `custom` build override

- QIRP SDK artifacts:

```
<workspace>/qcom-download-utils/qcom-6.6.65-QLI.
1.4-Ver.1.1/build-qcs9075-custom/tmp-glibc/
deploy/qirpsdk_artifacts/qcs9075-rb8-core-kit/
qirp-sdk_<version>.tar.gz
```

- Robotics image:

```
<workspace>/qcom-download-utils/qcom-6.6.65-QLI.
1.4-Ver.1.1/build-qcs9075-custom/tmp-glibc/
deploy/images/qcs9075-rb8-core-kit/qcom-
robotics-full-image
```

Example using `base` build override

- QIRP SDK artifacts:

```
<workspace>/qcom-download-utils/qcom-6.6.65-QLI.1.4-
Ver.1.1/build-qcs9075-base/tmp-glibc/deploy/qirpsdk_
artifacts/qcs9075-rb8-core-kit/qirp-sdk_<version>.tar.
gz
```

- Robotics image:

```
<workspace>/qcom-download-utils/qcom-6.6.65-QLI.1.4-
Ver.1.1/build-qcs9075-base/tmp-glibc/deploy/images/
qcs9075-rb8-core-kit/qcom-robotics-full-image
```

IQ-8 Beta Evaluation Kit

Example using `custom` build override

- QIRP SDK artifacts:

```
<workspace>/build-qcs8300-custom/tmp-glibc/
deploy/qirpsdk_artifacts/qcs8300-ride-sx/qirp-
sdk_<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs8300-custom/tmp-glibc/
deploy/images/qcs8300-ride-sx/qcom-robotics-
full-image
```

Example using `base` build override

- QIRP SDK artifacts:

```
<workspace>/build-qcs8300-base/tmp-glibc/deploy/
qirpsdk_artifacts/qcs8300-ride-sx/qirp-sdk_
<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs8300-base/tmp-glibc/deploy/
images/qcs8300-ride-sx/qcom-robotics-full-image
```

Flash the QIRP to devices

Flash the robotics image to the device, and install the QIRP SDK.

- To flash the robotics image to the device, see Flash software images.
- To install the QIRP SDK on the device, see Install the QIRP SDK on the device.

Build with GitHub (firmware and extras)

Use the detailed instructions to sync and build Qualcomm Linux selected firmware sources and build the QIRP SDK with its extra layers.

Prerequisite

The host environment is setup according to Ubuntu host setup in GitHub workflow (firmware and extras).

Steps:

1. Download the Qualcomm Yocto and QIRP SDK layers using either manifest or `git clone`.

   - **Download with manifest**: download the layers for the QIRP SDK by running the following commands.

     Note: To get the latest `<robotics-release-manifest>`, see the Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.

     ```
     cd <workspace>
     repo init -u https://github.com/quic-yocto/qcom-manifest -b
     qcom-linux-scarthgap -m <robotics-release-manifest>
     repo sync -c -j8
     ```

     Example:

     The following command downloads the release with manifest `qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-product-sdk-1.1.xml`:

     ```
     repo init -u https://github.com/quic-yocto/qcom-manifest -b
     qcom-linux-scarthgap -m qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-
     product-sdk-1.1.xml
     repo sync -c -j8
     ```

   - **Download with** `git clone`:

     a. Set up the host environment and sync the latest Yocto project BSP as described in the Qualcomm Repo Manifest README file.

     b. Download the layers for the QIRP SDK based on the `<workspace>` directory of the downloaded Yocto project BSP.

     ```
     cd <workspace>
     git clone https://github.com/ros/meta-ros -b scarthgap
     layers/meta-ros && cd layers/meta-ros && git checkout
     c560699e810e60a9526f4226c2c23f8d877280c8 && cd ../../
     git clone https://github.com/quic-yocto/meta-qcom-
     robotics.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-
     product-sdk-1.1 layers/meta-qcom-robotics
     git clone https://github.com/quic-yocto/meta-qcom-
     robotics-distro.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_
     robotics-product-sdk-1.1 layers/meta-qcom-robotics-distro
     git clone https://github.com/quic-yocto/meta-qcom-
     robotics-sdk.git -b qcom-6.6.65-QLI.1.4-Ver.1.1_robotics-
     product-sdk-1.1 layers/meta-qcom-robotics-sdk
     git clone https://github.com/quic-yocto/meta-qcom-qim-
     ```

```
product-sdk -b qcom-6.6.65-QLI.1.4-Ver.1.1_qim-product-
sdk-1.1.2 layers/meta-qcom-qim-product-sdk
```

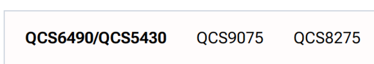
Results: The following layers are downloaded.

```
meta-qcom
meta-qcom-hwe
meta-qcom-distro
meta-ros
meta-qcom-robotics
meta-qcom-robotics-distro
meta-qcom-robotics-sdk
meta-qcom-qim-product-sdk
```

2. Sync and build the Qualcomm® Linux firmware.

   a. Sync the Qualcomm Linux firmware with the steps in Sync firmware, using the `firmware release tag` in Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.

   b. Build the Qualcomm Linux firmware with the steps in Build firmware



.

Note:

- For Qualcomm Dragonwing™ RB3 Gen 2 Vision Development Kit, use the steps under the tab **QCS6490/QCS5430**.
- For Qualcomm Dragonwing™ IQ-9075 Evaluation Kit, use the steps under the tab **QCS9075**.
- For Qualcomm® IQ-8 Beta Evaluation Kit, use the steps under the tab **QCS8275**.

Results:

For the preceding build, firmware prebuild is successful if the following zip files appear:

RB3 Gen 2 Vision Development Kit

File path: <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk/QCM6490.LE.1.0/common/build/ufs/bin

- QCM6490_bootbinaries.zip
- QCM6490_dspso.zip
- QCM6490_fw.zip

IQ-9075 Evaluation Kit

File path: <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk/QCS9100.LE.1.0/common/build/ufs/bin

- QCS9100_bootbinaries.zip
- QCS9100_dspso.zip
- QCS9100_fw.zip

IQ-8 Beta Evaluation Kit

File path: <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk/QCS8300.LE.1.0/common/build/ufs/bin

- QCS8300_bootbinaries.zip
- QCS8300_dspso.zip
- QCS8300_fw.zip

3. Fetch the meta-qcom-robotics-extras and meta-qcom-extras layers.

Note:

- To get the <meta-qcom-robotics-extras-release-tag>, see Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes. For example, r1.0_00077.0.
- To get the <product> value, see the following table.

Table: `product` value for extra layers

Parameter	RB3 Gen 2 Vision Development Kit	IQ-9075 Evaluation Kit	IQ-8 Beta Evaluation Kit
product	QCM6490.LE.1.0	QCS9100.LE.1.0	QCS8300.LE.1.0

```
git clone -b <meta-qcom-robotics-extras-release-tag> --depth 1
https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-
spf-1-0_hlos_oem_metadata.git
mkdir -p layers/meta-qcom-robotics-extras
mkdir -p layers/meta-qcom-extras
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/<product>/
common/config/meta-qcom-extras/* layers/meta-qcom-extras/
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/<product>/
common/config/meta-qcom-robotics-extras/* layers/meta-qcom-
robotics-extras/
```

4. Set up the build environment.

Note: If you are building the QIRP SDK on the Ubuntu Server VM of an Arm64 Mac, add the variable setting `SDKMACHINE=aarch64` in the following setup command.

```
MACHINE=<Machine_name> DISTRO=<Distro_name>  SDKMACHINE=aarch64
QCOM_SELECTED_BSP=<Build_override> source setup-robotics-
environment <Build_directory>
```

```
cd <workspace>
export CUST_ID="213195"
export FWZIP_PATH="<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_
standard_oem_nm-qirpsdk/<product>/common/build/ufs/bin"
ln -s layers/meta-qcom-robotics-distro/set_bb_env.sh ./setup-
robotics-environment
ln -s layers/meta-qcom-robotics-sdk/scripts/qirp-build ./qirp-
build
MACHINE=<Machine_name> DISTRO=<Distro_name>  QCOM_SELECTED_BSP=
<Build_override> source setup-robotics-environment <Build_
directory>
cat >> conf/bblayers.conf <<EOF
EXTRALAYERS = " \\
\${WORKSPACE}/layers/meta-qcom-robotics-extras \\
\${WORKSPACE}/layers/meta-qcom-extras \\
"
```

```
EOF
```

Table : Firmware specific parameters

Parameter	RB3 Gen 2 Vision Development Kit	IQ-9075 Evaluation Kit	IQ-8 Beta Evaluation Kit
FIRMWARE_ROOT	Root path of the firmware code	Root path of the firmware code	Root path of the firmware code
product	QCM6490.LE.1.0	QCS9100.LE.1.0	QCS8300.LE.1.0

Table : Build parameters

Parameter	RB3 Gen 2 Vision Kit	IQ-9075 Evaluation Kit	IQ-8 Beta Evaluation Kit
Machine name	qcs6490-rb3gen2-vision-kit	qcs9075-rb8-core-kit	qcs8300-ride-sx
Distro_name	qcom-robotics-ros2-jazzy	qcom-robotics-ros2-jazzy	qcom-robotics-ros2-jazzy
Build_override	custom	custom	custom
Build_directory	build-qcs6490-custom	build-qcs9075-custom	build-qcs8300-custom

5. Build the robotics image and QIRP SDK artifacts.

```
../qirp-build qcom-robotics-full-image
```

Results for different machines:

RB3 Gen 2 Vision Development Kit

- QIRP SDK artifacts:

```
<workspace>/build-qcs6490-custom/tmp-glibc/deploy/qirpsdk_
artifacts/qcs6490-rb3gen2-vision-kit/qirp-sdk_<version>.tar.
gz
```

- Robotics image:

```
<workspace>/build-qcs6490-custom/tmp-glibc/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-robotics-full-image
```

IQ-9075 Evaluation Kit

Example using machine_name `qcs9075-rb8-core-kit` and build_override `custom`

- QIRP SDK artifacts:

```
<workspace>/build-qcs9075-custom/tmp-glibc/deploy/qirpsdk_
artifacts/qcs9075-rb8-core-kit/qirp-sdk_<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs9075-custom/tmp-glibc/deploy/images/
qcs9075-rb8-core-kit/qcom-robotics-full-image
```

IQ-8 Beta Evaluation Kit

Example using machine_name `qcs8300-ride-sx` and build_override `custom`

- QIRP SDK artifacts:

```
<workspace>/build-qcs8300-custom/tmp-glibc/deploy/qirpsdk_
artifacts/qcs8300-ride-sx/qirp-sdk_<version>.tar.gz
```

- Robotics image:

```
<workspace>/build-qcs8300-custom/tmp-glibc/deploy/images/
qcs8300-ride-sx/qcom-robotics-full-image
```

Next steps

- Flash and install

Build the robotics image with the prebuilt robotics eSDK

Use the prebuilt platform extended SDK (eSDK) to build the robotics image.

The robotics eSDK is an installer generated from the Qualcomm Linux image and provides a complete Yocto environment that allows you to synchronize, modify, compile, and install applications.

Build the robotics image

Prerequisites:

- An Ubuntu 22.04 host computer with at least 50 GB of free space.
- You have downloaded the prebuilt robotics image with steps in Download and use the prebuilt package.

Steps:

1. Install the eSDK by running the installer script.

   a. Run the installer script with the following commands:

   RB3 Gen 2 Vision Development Kit

   ```
   cd <decompressed_workspace>/target/qcs6490-rb3gen2-vision-kit/sdk
   umask a+rx
   sh ./qcom-robotics-ros2-jazzy-x86_64-qcom-robotics-full-image-armv8-2a-qcs6490-rb3gen2-vision-kit-toolchain-ext-2.2.0.sh
   ```

IQ-9075 Evaluation Kit

```
cd <decompressed_workspace>/target/qcs9075-rb8-core-kit/sdk
umask a+rx
sh ./qcom-robotics-ros2-jazzy-x86_64-qcom-robotics-full-
image-armv8-2a-qcs9075-rb8-core-kit-toolchain-ext-2.2.0.sh
```

b. When you see the following prompt, press **Enter** or type a custom directory for eSDK installation.

```
QCOM Robotics Reference Distro with ROS Extensible SDK installer version 2.2.0

=============================================================================
Enter target directory for SDK (default: ~/qcom-robotics-ros2-jazzy_sdk):
```

2. Follow the instructions on the console to install the Platform eSDK in a convenient location of your host PC.

3. Ensure that the eSDK installation is successful when you see the following prompt.

```
SDK has been successfully set up and is ready to be used.
Each time you wish to use the SDK in a new shell session, you need to source the environment setup script.
```

4. Set up the eSDK and build the robotics image.

```
. environment-setup-armv8-2a-qcom-linux
devtool build-image qcom-robotics-full-image
```

Output:

RB3 Gen 2 Vision Development Kit

```
<eSDK_install_path>/tmp/deploy/images/
qcs6490-rb3gen2-vision-kit/qcom-robotics-full-image
```

IQ-9075 Evaluation Kit

```
<eSDK_install_path>/tmp/deploy/images/qcs9075-rb8-core-kit/
qcom-robotics-full-image
```

Note: The `<eSDK_install_path>` is the default or custom path specified in Step 1.

Develop with the eSDK

To develop your own application with the eSDK, see the Yocto documentation: Using devtool in your SDK workflow.

4.3 QIRP SDK folder structure

The QIRP SDK folder layout and description of the files and resources available to the users.

QIRP SDK layout

When decompressed, the QIRP SDK folder is organized as follows:

```
tree qirp-sdk -L 1
├── runtime
├── setup.sh
├── qirp-samples
└── toolchain
```

Name	Description
`runtime`	This directory contains the runtime environment necessary for executing applications developed with the QIRP SDK. It typically includes libraries and binaries that support the running of the applications on the target platform.
`setup.sh`	This shell script sets up the development environment for the QIRP SDK. Running this script usually configures the necessary environment variables and paths to ensure that the SDK's libraries are correctly accessible for application development and compilation.
`qirp-samples`	This directory houses example applications and code snippets provided as part of the QIRP SDK.

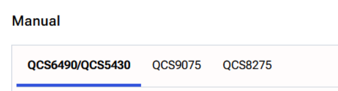
Name	Description
`toolchain`	This directory has the cross-compilation toolchain included in the QIRP SDK. The toolchain includes compilers, linkers, and other utilities needed to build applications for the target platform. This toolchain allows developers to compile their applications on a development machine (host) for execution on the target device (target).

4.4 Flash and install

Describes the procedure for flashing the robotics image and installing the QIRP SDK to the device.

Flash the robotics image

Steps:

1. Move the machine into EDL mode, using the manual steps 

in Move to EDL mode.

    - For Qualcomm Dragonwing™ RB3 Gen 2 Vision Development Kit, follow the steps under **QCS6940/QCS5430**.
    - For Qualcomm Dragonwing™ IQ-9075 Evaluation Kit, follow the steps under **QCS9075**.
    - For Qualcomm® IQ-8 Beta Evaluation Kit, follow the steps under **QCS8275**.

2. To flash one of the following robotics images to the device, follow the steps in Flash images.

    - Robotics image generated in Build with QSC-CLI
    - Robotics image generated in Build with GitHub
    - Robotics image generated in Build with GitHub (firmware and extras)

Flash on the Windows host

If you use the Windows Subsystem Linux (WSL) to build the robotics image, flash the image on the Windows host with the following steps:

Note: For WSL setup, see Set up an Ubuntu VM on Windows.

---

1. Install Microsoft WinUSB.

    a. Uninstall any other drivers for the device. Ensure that drivers like Qualcomm USB driver aren't installed. The device shouldn't appear under *COM Ports* in the *Device Manager*.

b. Plug your device into the host computer.

c. Open *Device Manager* and locate the device.

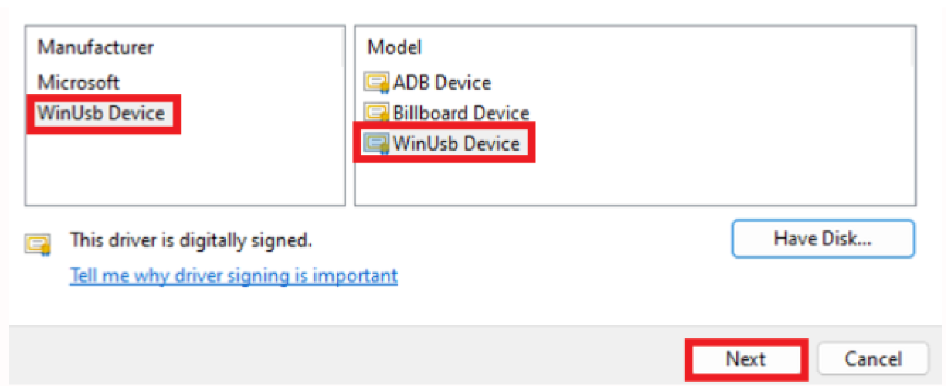
d. Right-click the device and select *Update driver software. . .* from the context menu.

e. In the wizard, select *Browse my computer for drivers*.

f. Select *Let me pick from a list of device drivers on my computer*.

g. From the list of device classes, select *Universal Serial Bus devices*.

h. The wizard displays *WinUsb Device*. Select it to load the driver.



2. Download the QDL tool and unzip the contents of the downloaded folder. Qualcomm Linux 1.4 requires QDL version 2.3.1 or higher.

3. To access the WSL workspace, go to the following path using the Windows File Explorer.

```
\\wsl$
```

4. Navigate to the path where the robotics image is generated as listed in the following topics.

   - Robotics image generated in Build with QSC-CLI
   - Robotics image generated in Build with GitHub
   - Robotics image generated in Build with GitHub (firmware and extras)

5. Copy the `QDL.exe` and `libusb-1.0.dll` from `\<qdl_root>\QDL_Win_x64` to the robotics image directory.

Note: Replace `qdl_root` (for example, 'qdl_2.3.4') with the actual directory name according to the version you download from the Qualcomm software center.

6. Type `powershell` in the Windows File Explorer's address bar, and press `Enter`. The power shell opens to the path of the robotics image.

7. To flash the images, run the following command:

```
.\QDL.exe prog_firehose_ddr.elf rawprogram0.xml rawprogram1.xml
rawprogram2.xml rawprogram3.xml rawprogram4.xml rawprogram5.xml
patch0.xml patch1.xml patch2.xml patch3.xml patch4.xml patch5.
xml
```

Install the QIRP SDK on the device

Steps:

1. On the host computer, move to the artifacts directory and decompress the package using the `tar` command.

```
cd <workspace>/build-qcom-robotics-ros2-jazzy\
/tmp-glibc/deploy/qirpsdk_artifacts
tar -zxf qirp-sdk_<qirp_version>.tar.gz
```

   The `qirp-sdk` directory is generated.

   Note: The `qirp-sdk_<qirp_version>.tar.gz` is in the deployed path of QIRP artifacts. The `<qirp_version>` changes with each release, such as `2.0.0`, `2.0.1`. For example, the whole package name can be `qirp-sdk_2.0.0.tar.gz`. For all released versions, see the Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Release Notes.

2. Enable `SSH` in ‘Permissive’ mode with the steps mentioned in Sign in using SSH.

3. To deploy the QIRP artifacts, push the QIRP files to the device using the following commands:

```
cd <workspace>/build-qcom-robotics-ros2-jazzy\
/tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk
scp ./runtime/qirp-sdk.tar.gz root@[ip-addr]:/opt/
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) cd /opt && tar -zxf ./qirp-sdk.tar.gz
(ssh) chmod +x /opt/scripts/*.sh
(ssh) cd /opt/scripts && ./install.sh
```

Next steps

You can now do the following:

- Develop a robotic application
- Check and run available sample applications

5 Develop a robotic application

The following example provides a general procedure for developing a ROS application using the QIRP SDK, using a ROS2 demo application on GitHub as an example.

Prerequisites:

- **Required for all cases**

  You have enabled `SSH` in ‘Permissive’ mode with the steps mentioned in Sign in using SSH.

- **Required when using the prebuilt package**
  - The prebuilt robotics image is flashed, see Flash and install.
  - The prebuilt QIRP SDK is downloaded.

Steps:

1. Set up the cross-compile environment.

   ```
   cd <qirp_decompressed_workspace>/qirp-sdk
   source setup.sh
   ```

2. Fetch the project and write your own code.

   a. Fetch a project from GitHub.

   ```
   git clone https://github.com/ros2/demos.git -b jazzy
   cd demos/demo_nodes_cpp
   vim src/topics/talker.cpp
   ```

   b. Develop your own application. The following is a sample.

   Change the `demo_nodes_cpp/src/topics/talker.cpp` msg data in line46, such as changing ‘Hello world’ to ‘get message success’

   ```
   46:msg_->data = "get message success " + std::to_
   string(count_++);
   ```

3. Compile the application.

```
colcon build --merge-install --cmake-args \
 -DPython3_NumPy_INCLUDE_DIR=${OECORE_TARGET_SYSROOT}/usr/lib/
python3.10/site-packages/numpy/core/include \
 -DCMAKE_STAGING_PREFIX=$(pwd)/install \
 -DCMAKE_PREFIX_PATH=$(pwd)/install/share \
 -DBUILD_TESTING=OFF \
 --packages-up-to demo_nodes_cpp
```

4. Push the demo to the device.

```
cd demo_nodes_cpp/install
tar -czvf demo_nodes_cpp.tar.gz lib share
scp demo_nodes_cpp.tar.gz root@[ip-addr]:/opt/
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) tar --no-same-owner -zxf /opt/demo_nodes_cpp.tar.gz -C /
usr/
```

5. Run the demo application on the device.

```
(ssh) export HOME=/opt
(ssh) source /usr/bin/ros_setup.sh && source /usr/share/qirp-
setup.sh
(ssh shell 1) ros2 run demo_nodes_cpp talker
(ssh shell 2) ros2 run demo_nodes_cpp listener
```

6 QIRP SDK sample applications

The QIRP SDK offers sample applications that you can run to experience basic functionality on the device. For example, the System Monitor ROS node publishes system information using ROS messages, such as CPU loading, memory usage, and battery status. Based on those sample applications, you can write your own Robotics/ROS applications.

Note: Some samples provided by QIRP SDK require a chassis of a mobile robot activated during testing, or the keyboard to control the movement of the robot. To use those samples, you must have your own mobile robot.

Table: QIRP SDK samp

Sample	Peripherals required	Mobile robot required	Support for RB3 Gen 2 Vision Kit	Support fo
			Custom	Custom
Orbbec-camera	Gemini 335L	N	Y	Y
Rplidar-ros2	RPLIDAR A3M1	N	Y	Y
Qrb-ros-imu	—	N	Y	N

Sample	Peripherals required	Mobile robot required	Support for RB3 Gen 2 Vision Kit	Support for I
			Custom	Custom
Qrb-ros-system-monitor	—	N	Y	Y
Ocr-service	—	N	Y	Y
QRB-ros-color-space-convert	—	N	Y	N

Note: You can find the `README` files in the sample applications directory.

6.1 Rplidar-ros2

The `rplidar-ros2` sample application provides basic device handling for the 2D laser scanner RPLIDAR A1/A2/A3/S1/S2/S3.

The figure shows the pipeline of the rplidar-ros2 sample. The sample gets the driver data from `rplidar` SDK, and then publishes the `/scan` topic.

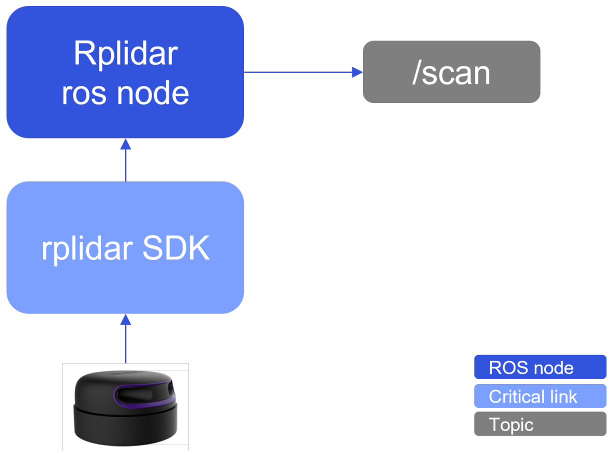


Figure : rplidar-ros2 pipeline

For information about the ROS nodes and topics used in the pipeline flow, see Pipeline flow for rplidar-ros2.

Use case: Run out-of-the-box rplidar-ros2

Prerequisites:

- You have built and installed QIRP SDK and dependencies, see Build the software and Flash and install.
- You have enabled `SSH` in 'Permissive' mode with the steps mentioned in Sign in using SSH.
- You have set up RPLIDAR A3M1.

Steps:

1. Start a terminal and run the following commands to set up QIRP SDK and ROS2 environment on the device and run rplidar-ros2.

```
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) source /usr/bin/ros_setup.bash
(ssh) ros2 launch rplidar_ros rplidar_a3_launch.py
```

Pipeline flow for rplidar-ros2

Table : ROS topic used in rplidar-ros2 pipeline

ROS topic	Type	Published by
`/scan`	`<sensor_msgs::msg::LaserScan>`	`rplidar-ros2` node

6.2 Qrb-ros-imu

The `Qrb-ros-imu` sample application creates the `/imu` topic to publish the inertial measurement unit (IMU) data.

The figure shows the pipeline, which captures IMU data from the IMU hardware and publishes the ROS topic messages.

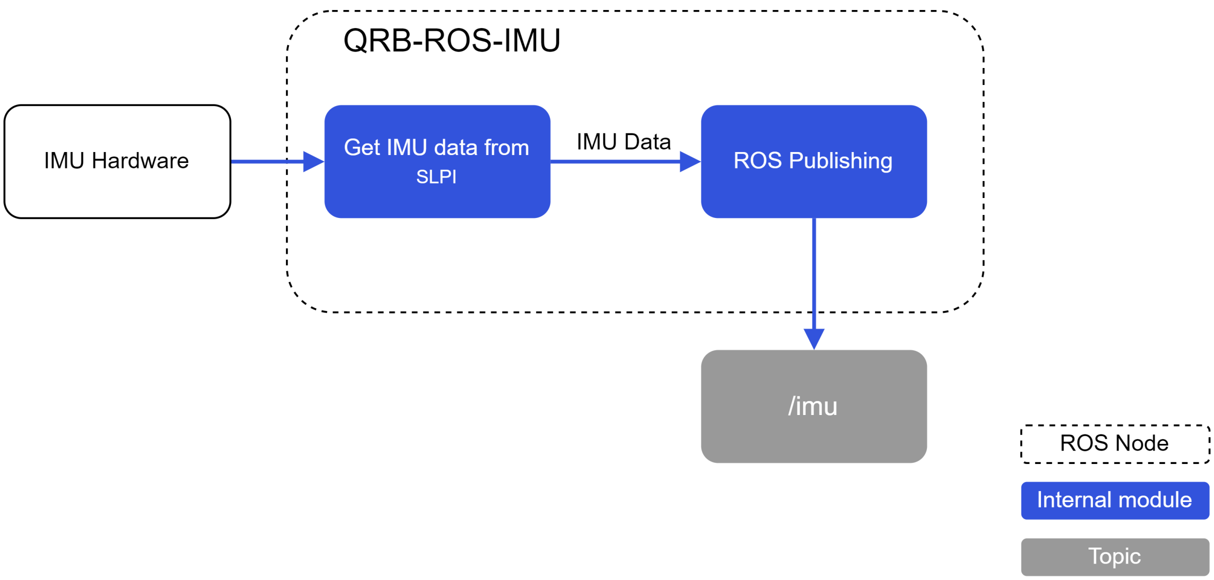


Figure : QRB-ROS-IMU pipeline

For information about the ROS nodes and topics used in the pipeline flow, see Pipeline flow for QRB-ROS-IMU.

Use cases

Prerequisites:

- You have built and installed the QIRP SDK and dependencies, see Build the software and Flash and install.
- You have enabled `SSH` in ‘Permissive’ mode with the steps mentioned in Sign in using SSH.

Case 1: Run out-of-the-box qrb-ros-imu

Steps:

1. Start two terminals, and run the following commands in each terminal to set up QIRP SDK and ROS2 environment on the device.

   Value range of `ROS_DOMAIN_ID`: [0, 232]

   ```
   ssh root@[ip-addr]
   (ssh) mount -o remount,rw /usr
   (ssh) export HOME=/opt
   (ssh) source /usr/share/qirp-setup.sh
   (ssh) export ROS_DOMAIN_ID=xx
   (ssh) source /usr/bin/ros_setup.bash
   ```

2. In terminal 1, run the IMU ROS Node.

   ```
   (ssh) ros2 run qrb_ros_imu imu_node
   ```

3. In terminal 2, get the IMU data.

   ```
   (ssh) ros2 topic echo /imu
   ```

Case 2: Build and run qrb-ros-imu

Steps:

1. Build `qrb_ros_imu` provided by the QIRP SDK on the host.

   ```
   cd [QIRP SDK path]
   source setup.sh

   cd qirp-samples/demos/platform/qrb_ros_imu
   colcon build --continue-on-error --cmake-args \
      -DCMAKE_TOOLCHAIN_FILE=${OE_CMAKE_TOOLCHAIN_FILE} \
      -DPYTHON_EXECUTABLE=${OECORE_NATIVE_SYSROOT}/usr/bin/python3
   \
      -DPython3_NumPy_INCLUDE_DIR=${OECORE_NATIVE_SYSROOT}/usr/lib/
   python3.12/site-packages/numpy/core/include \
   ```

```
   -DCMAKE_MAKE_PROGRAM=/usr/bin/make \
   -DBUILD_TESTING=OFF
```

2. Push `qrb_ros_imu` to the device.

Note: Ensure that the QIRP SDK is installed on the device.

```
cd qirp-samples/demos/platform/qrb_ros_imu/install/qrb_ros_imu
tar czvf qrb_ros_imu.tar.gz include lib share
scp qrb_ros_imu.tar.gz root@[ip-addr]:/opt
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) tar --no-same-owner -zxf /opt/qrb_ros_imu.tar.gz -C /usr/
```

3. Run `qrb_ros_imu` on the device by referring to Case 1: Run out-of-the-box qrb-ros-imu.

Pipeline flow for QRB-ROS-IMU

Table : ROS node used in qrb-ros-imu pipeline

ROS node	Description
`IMU`	Publishes the `/imu` ROS topic.

Table : ROS topic used in qrb-ros-imu pipeline

ROS topic	Type	Published by
`/ros`	`<sensor_msgs::msg::IMU>`	`IMU` node

Limitation

The frame rate can only be fixed values as it's limited by hardware. Custom settings aren't supported.

6.3 Qrb-ros-system-monitor

The `Qrb-ros-system-monitor` sample application has various ROS nodes that publish system status information, such as CPU loading, memory usage, and disk spaces.

The figure shows the pipeline, which collects system information with `sysfs` node and some system standard command tools, then publishes these data with ROS messages.

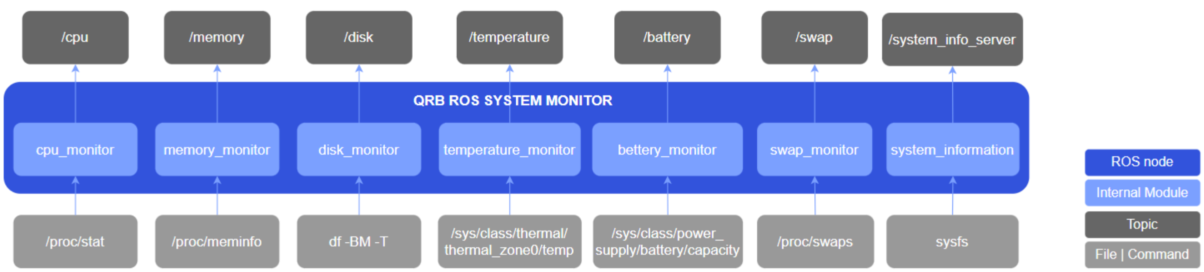


Figure : qrb-ros-system-monitor pipeline

For information about the ROS nodes and topics used in the pipeline flow, see Pipeline flow for QRB-ROS-SYSTEM-MONITOR.

Use Cases

Prerequisites:

- You have built and installed the QIRP SDK and dependencies, see Build the software and Flash and install.
- You have enabled `SSH` in 'Permissive' mode with the steps mentioned in Sign in using SSH.

Case 1: Run out-of-the-box qrb-ros-system-monitor

Steps:

1. Start two terminals, and run the following commands in each terminal to set up QIRP SDK and ROS2 environment on the device.

   Value range of `ROS_DOMAIN_ID`: [0, 232]

   ```
   ssh root@[ip-addr]
   (ssh) mount -o remount,rw /usr
   (ssh) export HOME=/opt
   ```

```
(ssh) source /usr/share/qirp-setup.sh
(ssh) export ROS_DOMAIN_ID=xx
(ssh) source /usr/bin/ros_setup.bash
```

2. In terminal 1, run the system monitor ROS Node.

```
(ssh) ros2 run qrb_ros_system_monitor qrb_ros_system_monitor
```

3. In terminal 2, check the system information with ROS topic.

```
(ssh) ros2 topic echo /cpu
```

Case 2: Build and run qrb-ros-system-monitor

Steps:

1. Build `qrb_ros_system_monitor` provided by the QIRP SDK on the host.

```
cd <qirp_decompressed_workspace>
source setup.sh

cd qirp-samples/demos/platform/qrb_ros_system_monitor
colcon build --merge-install --cmake-args \
  -DCMAKE_TOOLCHAIN_FILE=${OE_CMAKE_TOOLCHAIN_FILE} \
  -DPYTHON_EXECUTABLE=${OECORE_NATIVE_SYSROOT}/usr/bin/python3 \
  -DPython3_NumPy_INCLUDE_DIR=${OECORE_NATIVE_SYSROOT}/usr/lib/
python3.12/site-packages/numpy/core/include \
  -DPYTHON_SOABI=cpython-312-aarch64-linux-gnu \
  -DCMAKE_MAKE_PROGRAM=/usr/bin/make \
  -DCMAKE_LIBRARY_PATH=${OECORE_TARGET_SYSROOT}/usr/lib \
  -DBUILD_TESTING=OFF
```

2. Push `qrb_ros_system_monitor` to the device.

Note: Ensure that the QIRP SDK is installed on the device.

```
cd qirp-samples/demos/platform/qrb_ros_system_monitor/install
tar czvf qrb_ros_system_monitor.tar.gz include lib share
scp qrb_ros_system_monitor.tar.gz root@[ip-addr]:/opt/
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) tar --no-same-owner -zxf /opt/qrb_ros_system_monitor.tar.
gz -C /usr/
```

3. Start two terminals, and run the following commands in each terminal to set up QIRP SDK and ROS2 environment on the device.

   Value range of `ROS_DOMAIN_ID`: [0, 232]

```
ssh root@[ip-addr]
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) export ROS_DOMAIN_ID=xx
(ssh) source /usr/bin/ros_setup.bash
```

4. In terminal 1, run the system monitor ROS node.

```
(ssh) ros2 run qrb_ros_system_monitor qrb_ros_system_monitor
```

5. In terminal 2, get the system information.

```
(ssh) ros2 topic echo /cpu
(ssh) ros2 topic echo /memory
```

ROS node	Description

Pipeline flow for QRB-ROS-SYSTEM-MONITOR

Table : ROS nodes used in qrb-ros-system-monitor pipeline

ROS node	Description
`CpuMonitor`	Publishes CPU utilization information.
`MemoryMonitor`	Publishes memory usage information.
`DiskMonitor`	Publishes disk utilization information.
`SwapMonitor`	Publishes swap space information
`TemperatureMonitor`	Publishes CPU temperature.
`BatteryMonitor`	Publishes battery information.
`SystemInfoServer`	Publishes static system information, such as CPU count.

Table : ROS topics used in qrb-ros-system-monitor pipeline

ROS topic	Type	Description
`/cpu`	`qrb_ros_system_monitor_interfaces/msg/CpuInfo`	CPU utilization information.

ROS topic	Type	Description
`/memory`	`qrb_ros_system_monitor_interfaces/msg/MemInfo`	Memory utilization information.
`/disk`	`qrb_ros_system_monitor_interfaces/msg/DiskInfo`	Disk utilization information.
`/swap`	`qrb_ros_system_monitor_interfaces/msg/SwapInfo`	Swap space information
`/temperature`	`std_msgs/msg/Float32`	CPU temperature.
`/battery`	`std_msgs/msg/Float32`	Battery information.

Table : ROS services used in qrb-ros-system-monitor pipeline

ROS service	Type	Description
`/system_info_server`	`qrb_ros_system_monitor_interfaces/srv/SystemInfo`	Static system information, such as CPU count.

Limitation

The `sysfs` node and Linux standard command-line tools collect the system information values. Therefore, if the `sysfs` path changes or command-line tools output format changes, those ROS nodes may not work as expected.

6.4 Ocr-service

The `ocr-service` sample application enables a service that provides the Optical Character Recognition (OCR) function.

It captures the image topic from the ROS system and publishes the result with the `ocr_topic`.

This figure shows the basic messages and data transfer channels, with the relevant client/server and ROS node.

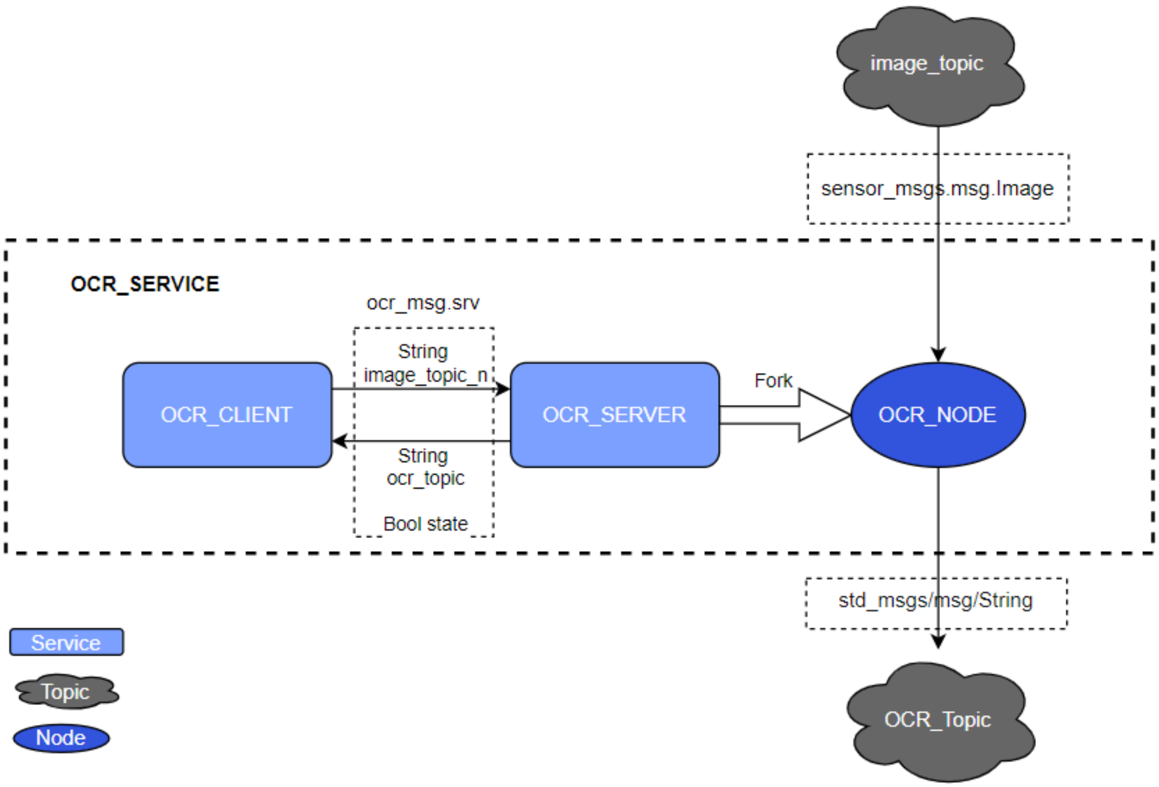


Figure : ocr-service pipeline

For information about the ROS nodes and topics used in the pipeline flow, see Pipeline flow for OCR-service .

Use cases

Prerequisites:

- You have built and installed the QIRP SDK and dependencies, see Build the software and Flash and install.
- You have enabled SSH in 'Permissive' mode with the steps mentioned in Sign in using SSH.

Case 1: Run out-of-the-box OCR-service

Steps:

1. Start three terminals, and run the following commands in each terminal to set up QIRP SDK and ROS2 environment on the device.

   Value range of ROS_DOMAIN_ID: [0, 232]

```
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) export ROS_DOMAIN_ID=xx
(ssh) source /usr/bin/ros_setup.bash
```

2. In terminal 1, launch ocr_server.

```
(ssh) ros2 run ocr_service ocr_server
```

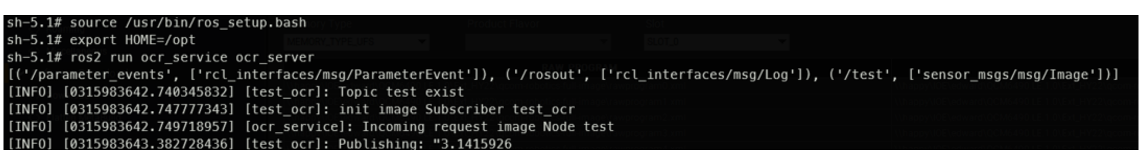
3. In terminal 2, push an image to the device and run the ocr_testnode.

```
scp  qirp-samples/demos/platform/ocr_service/scripts/digital_
720p.png root@[ip-addr]:/opt
(ssh) ros2 run ocr_service ocr_testnode --topic imagetest --
picture /opt/digital_720p.png
```

4. In terminal 3, run the ocr_client.

```
(ssh) ros2 run ocr_service ocr_client imagetest
```

   You will see the following log:



Case 2: Build and run OCR-service

Steps:

1. Build `ocr_msg and ocr_service` provided by the QIRP SDK on the host.

```
cd <qirp_decompressed_workspace>
source setup.sh
cd qirp-samples/demos/platform/ocr-service
colcon build --merge-install --cmake-args \
  -DPython3_NumPy_INCLUDE_DIR=${Python3_NumPy_INCLUDE_DIR} \
  -DPYTHON_SOABI=cpython-312-aarch64-linux-gnu \
  -DCMAKE_STAGING_PREFIX=$(pwd)/install \
  -DCMAKE_PREFIX_PATH=$(pwd)/install/share \
  -DBUILD_TESTING=OFF \
  -DCMAKE_MAKE_PROGRAM=/usr/bin/make \
  -DPython3_NumPy_INCLUDE_DIR=${OECORE_NATIVE_SYSROOT}/usr/lib/
python3.12/site-packages/numpy/core/include
```

2. Push `ocr_msg and ocr_service` to the device.

```
cd qirp-samples/demos/platform/ocr-service/install
tar -zcvf ocr_service.tar.gz include/ lib/ share/
ssh root@[ip-addr]
(ssh) mount -o remount rw /
scp ocr_service.tar.gz root@[ip-addr]:/opt
ssh ssh root@[ip-addr]
(ssh) tar --no-same-owner -zxf /opt/ocr_service.tar.gz -C /usr/
scp  <your pciture> root@[ip-addr]:/opt
```

3. Run `ocr_service` on the device referring to Case 1: Run out-of-the-box OCR-service.

Pipeline flow for OCR-service

Table : ROS nodes used in OCR-service pipeline

ROS node	Description
`OCR_CLIENT`	Sends the OCR request to `ocr_server` and gets the server result.
`OCR_SERVER`	Gets the OCR request from `ocr_server` , and forks the `ocr_rosnode` according to the request.
`OCR_ROSNODE`	Subscribes to the image topic according to the request.

Table : ROS topics used in OCR-service pipeline

ROS topic	Type	Published by
`/image_topic_name`	`< sensor_msgs.msg. Image >`	Any ROS node
`/ocr_image_topic_name`	`< std_msgs/msg/String >`	`ocr-rosnode` node

6.5 Orbbec-camera

The `Orbbec-camera` sample application enables the Orbbec Gemini camera 335L to work in RGB or depth mode. This application generates the RGB and depth information by topics.

Orbbec-camera is the OrbbecSDK ROS2 Wrapper, which provides seamless integration of Orbbec cameras within the ROS 2 environment. The following figure shows the pipeline of `orbbec-camera`.

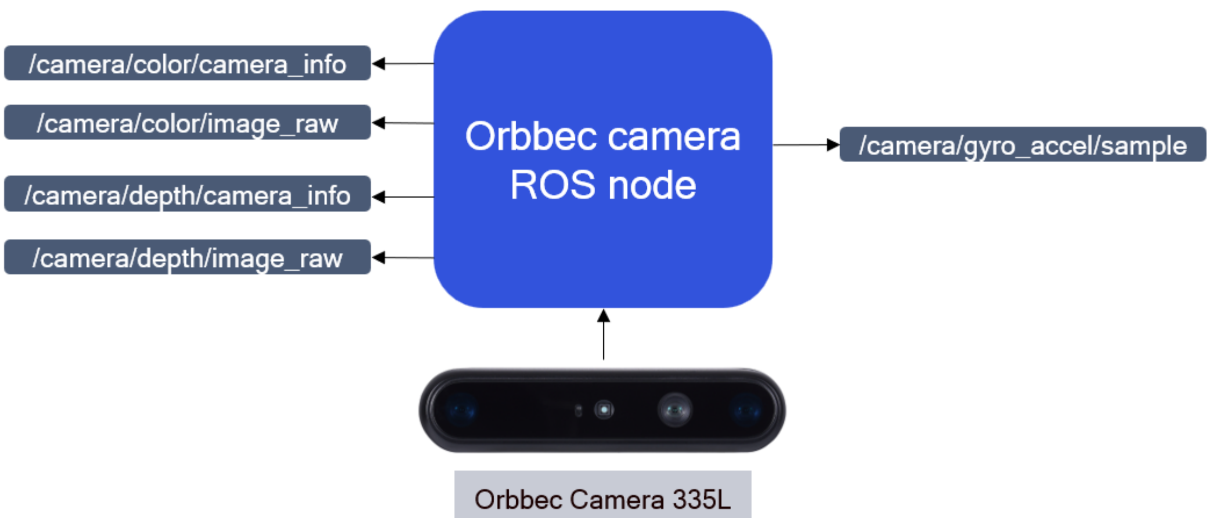


Figure : the orbbec-camera pipeline

For information about the ROS nodes and topics used in the pipeline flow, see Pipeline flow for orbbec-camera.

Use case: Run out-of-the-box orbbec-camera

Prerequisites:

- You have built and installed the QIRP SDK and dependencies, see Build the software and Flash and install.
- You have enabled `SSH` in 'Permissive' mode with the steps mentioned in Sign in using SSH.
- You have set up the Orbbec Gemini 335L camera.

Orbbec camera 335L supports the following 3 scenes:

1. To set up QIRP SDK and ROS2 environment on the device and start the `orbbec-camera` ROS node with the `RGB` module enabled, start a new terminal and run the following commands:

```
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) source /usr/bin/ros_setup.bash
(ssh) cd /usr/share/orbbec_camera/scripts/
(ssh) bash install_udev_rules.sh
(ssh) udevadm control --reload-rules && udevadm trigger
(ssh) source ../scripts/dds_config.sh
(ssh) ros2 launch orbbec_camera gemini_330_series.launch.py
depth_registration:=false enable_depth:=false enable_point_
cloud:=false color_width:=848 color_height:=480 color_fps:=30
color_qos:=default
```

2. To set up QIRP SDK and ROS2 environment on the device and start the `orbbec-camera` ROS node with the `Depth` module enabled, start a new terminal and run the following commands:

```
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) source /usr/bin/ros_setup.bash
(ssh) cd /usr/share/orbbec_camera/scripts/
(ssh) bash install_udev_rules.sh
(ssh) udevadm control --reload-rules && udevadm trigger
(ssh) ros2 launch orbbec_camera gemini_330_series.launch.py
depth_registration:=true enable_depth:=true enable_sync:=true
enable_point_cloud:=false color_width:=848 color_height:=480
color_fps:=30 depth_fps:=30
```

3. To set up QIRP SDK and ROS2 environment on the device and start the `orbbec-camera` ROS node with the `IMU` module enabled, start a new terminal and run the following commands:

```
(ssh) export HOME=/opt
(ssh) source /usr/share/qirp-setup.sh
(ssh) source /usr/bin/ros_setup.bash
(ssh) cd /usr/share/orbbec_camera/scripts/
(ssh) bash install_udev_rules.sh
(ssh) udevadm control --reload-rules && udevadm trigger
(ssh) ros2 launch orbbec_camera gemini_330_series.launch.py
depth_registration:=false enable_depth:=false enable_point_
cloud:=false color_width:=640 color_height:=360 color_fps:=30
color_qos:=default enable_accel:=true enable_gyro:=true enable_
sync_output_accel_gyro:=true
```

Pipeline flow for orbbec-camera

Table : ROS topics used in orbbec-camera pipeline

ROS topic	Type	Description
`/camera/color/camera_info`	`<sensor_msgs::msg::CameraInfo>`	Published by the `orbbec-camera` node.
`/camera/color/image_raw`	`<sensor_msgs::msg::Image>`	Published by the `orbbec-camera` node.
`/camera/depth/camera_info`	`<sensor_msgs::msg::CameraInfo>`	Published by the `orbbec-camera` node.
`/camera/depth/image_raw`	`<sensor_msgs::msg::Image>`	Published by the `orbbec-camera` node.
`/camera/gyro_accel/sample`	`<sensor_msgs::msg::Imu>`	Published by the `orbbec-camera` node.

6.6 Qrb-ros-color-space-convert

The `Qrb-ros-color-space-convert` sample application converts between NV12 and RGB888 formats.

Qualcomm's smart devices, such as the RB3 Gen 2, use NV12 as the default image color space conversion format. However, the more common color space format is RGB888. The `Qrb-ros-color-space-convert` sample application implements the following:

- Provides ROS nodes
  - API to convert nv12 to rgb8
  - API to convert rgb8 to nv12
- Supports `dmabuf` fd as input and output
- Input and output image receive and send with QRB ROS transport
- Hardware acceleration with GPU by OpenGL ES

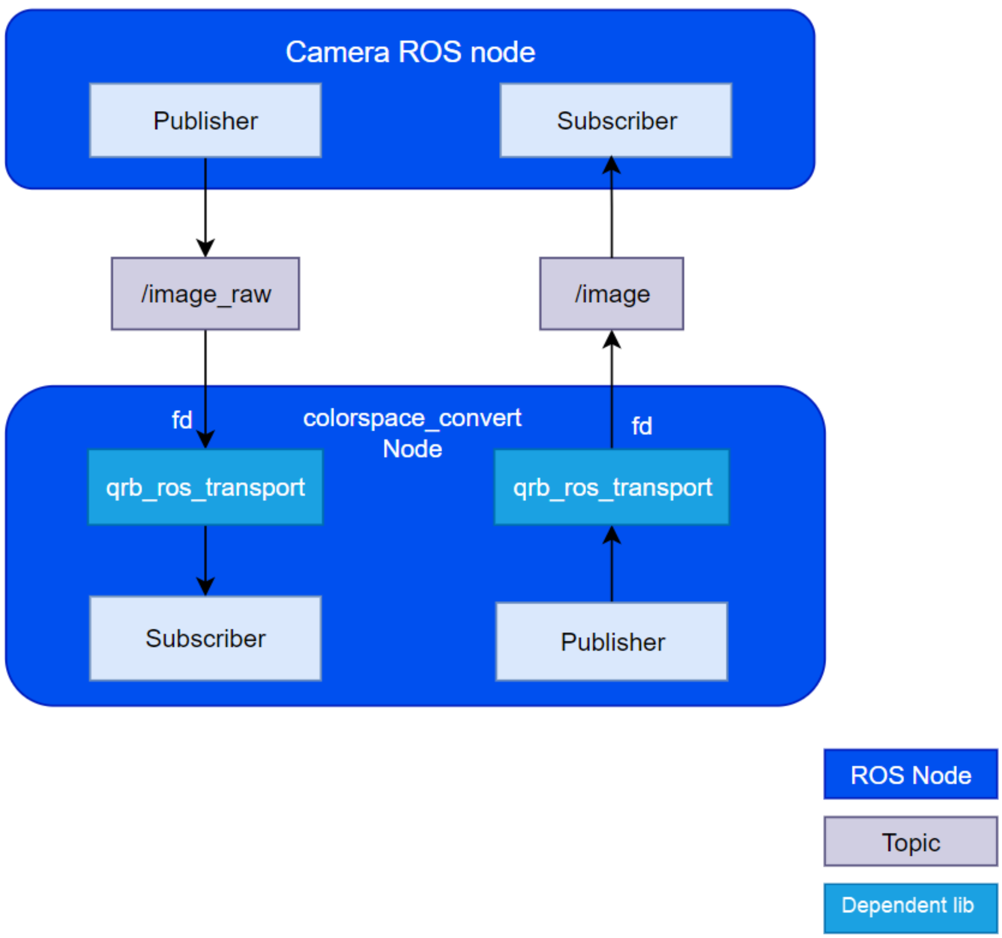


Figure : qrb-ros-color-space-convert pipeline

Use cases

Prerequisites:

- You have built and installed the QIRP SDK and dependencies, see Build the software and Flash and install.
- You have enabled `SSH` in 'Permissive' mode with the steps mentioned in Sign in using SSH.

Case 1: Run out-of-the-box qrb-ros-color-space-convert

Steps:

1. Write your own ROS node and run it in terminal.
2. In terminal 1, set up QIRP SDK and ROS2 environment on the device.
   a. set up environment.

   ```
   (ssh) export HOME=/opt
   (ssh) source /usr/share/qirp-setup.sh
   (ssh) export ROS_DOMAIN_ID=xx
   (ssh) source /usr/bin/ros_setup.bash
   (ssh) export XDG_RUNTIME_DIR=/dev/socket/weston/
   (ssh) mkdir -p $XDG_RUNTIME_DIR
   (ssh) export WAYLAND_DISPLAY=wayland-1
   (ssh) export FASTRTPS_DEFAULT_PROFILES_FILE=/usr/share/qrb_
   ros_colorspace_convert/config/large_message_profile.xml
   ```

   b. NV12 convert to RGB888, you can run the command

   ```
   (ssh) ros2 launch qrb_ros_colorspace_convert colorspace_
   convert.launch.py 'conversion_type:=nv12_to_rgb8' 'latency_
   fps_test:=True'
   ```

   c. RGB888 convert to NV12, you can run the command

   ```
   (ssh) ros2 launch qrb_ros_colorspace_convert colorspace_
   convert.launch.py 'conversion_type:=rgb8_to_nv12' 'latency_
   fps_test:=True'
   ```

Case 2: Build and run qrb-ros-color-space-convert

Steps:

1. Build `qrb_ros_color_space_convert` provided by the QIRP SDK on the host.

   ```
   cd <qirp_decompressed_workspace>
   source setup.sh
   ```

```
cd qirp-samples/demos/platform/qrb_ros_color_space_convert

colcon build --continue-on-error --cmake-args \
  -DCMAKE_TOOLCHAIN_FILE=${OE_CMAKE_TOOLCHAIN_FILE} \
  -DPYTHON_EXECUTABLE=${OECORE_NATIVE_SYSROOT}/usr/bin/python3 \
  -DPython3_NumPy_INCLUDE_DIR=${OECORE_NATIVE_SYSROOT}/usr/lib/
python3.10/site-packages/numpy/core/include \
  -DSYSROOT_LIBDIR=${OECORE_TARGET_SYSROOT}/usr/lib \
  -DSYSROOT_INCDIR=${OECORE_TARGET_SYSROOT}/usr/include \
  -DCMAKE_MAKE_PROGRAM=/usr/bin/make \
  -DBUILD_TESTING=OFF
```

2. Push `qrb_ros_color_space_convert` to the device.

Note: Ensure that the QIRP SDK is installed on the device.

```
cd qirp-samples/demos/platform/qrb_ros_color_space_convert/
install/qrb_ros_colorspace_convert
tar czvf qrb_ros_color_space_convert.tar.gz lib share
scp qrb_ros_color_space_convert.tar.gz root@[ip-addr]:/opt/
ssh root@[ip-addr]
(ssh) mount -o remount,rw /usr
(ssh) tar --no-same-owner -zxf /opt/qrb_ros_color_space_convert.
tar.gz -C /usr/
```

3. Run `qrb_ros_color_space_convert` on the device by referring to Case 1: Run out-of-the-box qrb-ros-color-space-convert.

Pipeline flow for Qrb-ros-color-space-convert

Table : ROS node used in qrb-ros-color-space-convert pipeline

ROS node	Description
`colorspace_convert_node`	Publishes the `/image` ROS topic.

Table : ROS topics used in qrb-ros-color-space-convert pipeline

ROS topic	Type	Published by
`/image`	`<qrb_ros::transport::type::Image>`	`qrb_ros_color_space_convert` node

7 Upgrade individual SDKs

QIRP SDK collects the libs/headers from different function SDKs. Currently QIRP SDK includes Qualcomm IM SDK and Qualcomm AI Engine Direct SDK (QNN). You can upgrade those SDKs individually in the following methods:

- Upgrade to the latest version
- Upgrade to a specified version by tags

Note: When changing the version of any function SDK to the latest, which isn't verified in QIRP SDK, conflicts or functionality issues can occur. Officially, the QIRP SDK supported versions are listed in the configuration file (JSON) to ensure the backward compatibility.

7.1 Upgrade the Qualcomm IM SDK

You can upgrade the version of Qualcomm IM SDK to either the latest or specified version by upgrading the Qualcomm Intelligent Multimedia Product (QIMP) SDK.

Upgrade Qualcomm IM SDK to the latest version

To upgrade the Qualcomm IM SDK to the latest version, follow these steps:

Steps:

1. Go to the `meta-qcom-qim-product-sdk` directory.

   ```
   cd <workspace>/layers/meta-qcom-qim-product-sdk
   ```

2. Get the latest version using git.

   ```
   git remote update
   git pull github scarthgap
   ```

3. After upgrade, recompile the QIRP SDK using the steps in Build with Dockerfile.

Upgrade Qualcomm IM SDK to a specific version by tags

To upgrade the Qualcomm IM SDK by tags, follow these steps:

Steps:

1. Find the `<release tag names>` of the `meta-qcom-qim-product-sdk` layer at https://github.com/quic-yocto/meta-qcom-qim-product-sdk/tags.
2. Go to the `meta-qcom-qim-product-sdk` directory.

```
cd <workspace>/layers/meta-qcom-qim-product-sdk
```

3. Upgrade to the specified release tag.

```
git remote update
git checkout -b <Release Tag Name>
```

4. After upgrade, recompile the QIRP SDK using the steps in Build with Dockerfile.

7.2 Upgrade the Qualcomm AI Engine Direct SDK (QNN)

You can upgrade the version of Qualcomm AI Engine Direct SDK (QNN) to a specified version.

To upgrade the Qualcomm AI Engine Direct SDK to a specified version, follow these steps:

Steps:

1. Find the `<release version>` of Qualcomm AI Engine Direct SDK on Qualcomm Software Center (QSC).
   a. Go to QSC, search and enter the **Qualcomm AI Engine Direct SDK** page.
   b. Choose **Linux** under **OS** and find a version from the **Version** dropdown list.
2. Download the required version SDK by running the following command:

   Note: Replace `${QNPSDK_SRC_VER}` with the required version.

```
wget https://softwarecenter.qualcomm.com/api/download/software/
qualcomm_neural_processing_sdk/v${QNPSDK_SRC_VER}.zip
```

   Example:

```
wget https://softwarecenter.qualcomm.com/api/download/software/
qualcomm_neural_processing_sdk/v2.22.0.240425.zip
```

3. Get the `sha256sum` value of the QNN zip file corresponding to the specified release version with this command:

```
sha256sum v${QNPSDK_SRC_VER}.zip
```

Example:

```
sha256sum v2.22.0.240425.zip
```

4. Update the following variables in the file `<workspace>/layers/meta-qcom-robotics-sdk/recipes-sdk/function-sdks/qti-qnn.bb`.

   a. Update `QNPSDK_SRC_VER="$release version "`.

      Example:

      ```
      QNPSDK_SRC_VER="2.22.0.240425"
      ```

   b. Update `QNPSDK_SRC_SHID` with the `sha256sum` value.

      Example:

```
QNPSDK_SRC_SHID="d68ed4d92187101a9759384cbce0a35bd383840b2e3c3c7
46a4d35f99823a75a"
```

5. After upgrade, recompile the QIRP SDK using the steps in Build with Dockerfile.

8 Troubleshooting

This troubleshooting information provides resolutions to common issues compiling and using the QIRP SDK.

Note: For common issues regarding Docker, sync, build, and flash, see Troubleshooting of the Qualcomm® Linux Build Guide.

9 Appendixes

Provides the reference information about the QIRP SDK.

9.1 Robotics layers specifications

Provides the detailed information for the robotics layers used in QIRP SDK.

The QIRP SDK includes both Qualcomm Linux layers and robotics layers. This information lists the details of all robotics layers.

- QIRP robotics layers
  - meta-ros
  - meta-qcom-robotics-sdk
  - meta-qcom-robotics
  - meta-qcom-robotics-distro
- Robotics extra layer

  meta-qcom-robotics-extras

`meta-ros`

The `meta-ros` layer information is published to the meta-ros GitHub repository.

`meta-qcom-robotics-sdk`

- **BitBake classes**

  The following table lists the BitBake classes defined in the meta data layer `meta-qcom-robotics-sdk`:

BitBake class	Description
`psdk-package.bbclass`	– Provides a packaging task to pack QIRP SDK artifacts into an archive. It's invoked by the `qirp-sdk` recipe. – The easy-to-install artifact archives are available at the `<workspace>/build-qcom-wayland/tmp-glibc/deploy/qirpsdk_artifacts` directory after the recipe build is complete.
`psdk-base.bbclass`	– Provides base configurations to set the package name and disable the unused tasks for QIRP SDK. – Invoked by the `qirp-sdk` recipe during the build.
`psdk-extract.bbclass`	– Provides an extracting task to decompress function SDKs artifacts to source directory with various compression formats. – Invoked by the `qti-qim-product-sdk/qti-robotics` recipe during the build.
`psdk-install.bbclass`	– Provides an installing task to split function SDKs artifacts with common format . the artifacts is available at <workspace>/build-qcom-wayland/tmp-glibc/deploy/artifacts/${PN}_artifacts/ – Invoked by the `qti-qim-product-sdk/qti-qnn/qti-robotics` recipe during the build.
`psdk-image.bbclass`	– Provides an qirpsdk generation task to collect each function sdk artifacts,trigger the do_populate_sdk task and collect the standard sdk for qirpsdk generation. – Invoked by the `qcom-robotics-full-image` recipe during the build.

BitBake class	Description
psdk-pickup.bbclass	– Provides a pickup task that selects files in function SDKs based on the configuration file (config_content.json by default), moves all selected files to the / directory, and finally integrates them into the QIRP SDK package. – Invoked by the qirp-sdk recipe during the build.

- **Distro configuration**

layer.conf	Configures the project layers with the following information: – Recipe file path – Supported Yocto version – Filename

- **Recipes**

Recipe	Description
recipes-sdk	Consists of function SDK recipes and QIRP SDK recipe: – qti-qim.bb – qti-qim-product-sdk.bb – qti-robotics.bb – qti-qnn.bb – qirp-sdk.bb

meta-qcom-robotics

- **BitBake classes**

The following table lists the BitBake classes defined in the metadata layer meta-qcom-robotics-sdk:

BitBake class	Description
fsdk-package.bbclass	• Provides a packaging task to pack the Robotics artifacts into an archive. It's invoked by the packagegroup-qcom-robotics recipe. • The easy-to-install artifact archives are available at the <workspace>/build-qcom-wayland/tmp-glibc/deploy/roboticssdk_artifacts directory after the recipe build is complete.
psdk-base.bbclass	• Provides base configurations to set the package name and disables the unused tasks for the Robotics artifacts. • Invoked by packagegroup-qcom-robotics recipe during the build.
robotics-package.bbclass	• Provides a task to move all robotics files to the pkg_dest directory. pkg_dest default with "/" • Provides base configurations to set the package name and package file. • Invoked by the robotics feature recipes during the build.

- **Distro configuration**

layer.conf	Configures the project layers with the following information: – Recipe file path – Supported Yocto version

- **Recipes**

Recipe	Description
recipes	**Consists of robotics feature recipes:** – `libqrc-udriver.bb` – `librealsense2_2.54.2.bb` – `mcb-flash_0.0.1.bb` – `nuttx_0.0.1.bb` – `nuttx-apps_0.0.1.bb` – `ncnn.bb` – `orbbec-camera_1.5.10-1.bb` – `orbbec-camera-msgs_1.2.2-1.bb` – `orbbec-description_0.0.0-1.bb` – `packagegroup-qti-robotics.bb` – `ranger-mini-base_0.0.1.bb` – `ranger-mini-bringup_0.0.1.bb` – `ranger-mini-msg_0.0.1.bb` – `ugv-sdk_0.0.1.bb` – `battery-client.bb` – `battery-service_0.1.bb` – `qrb-ros-battery_0.1.bb` – `qrb-ros-camera_0.1.bb` – `qti-robot-amr-ctrl.bb` – `qti-robot-keyboard.bb` – `qti-robot-urdf.bb` – `qrb-ros-imu_1.0.bb` – `sensor-client.bb` – `rplidar-ros2_2.1.2-1.bb` – `ocr-msg.bb` – `ocr-service.bb` – `python3-pytesseract_0.3.10.bb`
recipes-sdk	Consists of scripts and packagegroup-qcom-robotics recipe: – `packagegroup-qcom-robotics.bb`

Recipe	Description
recipes-bbappends	Consists of the recipe append files, which add extended configuration: – `ceres-solver_%.bbappend` – `nav2-bringup_1.1.5-1.bbappend` – `navigation2_%.bbappend` – `python3-lark-parser_0.7.0.bbappend` – `python3-pybind11_2.11.1.bbappend` – `realsense2-camera_4.51.1-1.bbappend` – `realsense2-camera-msgs_4.51.1-1.bbappend` – `backward-ros_1.0.5-1.bbappend` – `diagnostic-updater_4.2.1-1.bbappend` – `dwb-core_1.3.2-1.bbappend` – `dwb-critics_1.3.2-1.bbappend` – `dwb-msgs_1.3.2-1.bbappend` – `dwb-plugins_1.3.2-1.bbappend` – `nav2-amcl_1.3.2-1.bbappend` – `nav2-behaviors_1.3.2-1.bbappend` – `nav2-bt-navigator_1.3.2-1.bbappend` – `nav2-collision-monitor_1.3.2-1.bbappend` – `nav2-constrained-smoother_1.3.2-1.bbappend` – `nav2-controller_1.3.2-1.bbappend` – `nav2-costmap-2d_1.3.2-1.bbappend` – `nav-2d-msgs_1.3.2-1.bbappend` – `nav-2d-utils_1.3.2-1.bbappend` – `nav2-graceful-controller_1.3.2-1.bbappend` – `nav2-lifecycle-manager_1.3.2-1.bbappend` – `nav2-map-server_1.3.2-1.bbappend` – `nav2-msgs_1.3.2-1.bbappend` – `nav2-navfn-planner_1.3.2-1.bbappend` – `nav2-planner_1.3.2-1.bbappend` – `nav2-regulated-pure-pursuit-controller_1.3.2-1.bbappend` – `nav2-rotation-shim-controller_1.3.2-1.bbappend` – `nav2-smoother_1.3.2-1.bbappend` – `nav2-theta-star-planner_1.3.2-1.bbappend` – `nav2-util_1.3.2-1.bbappend` – `nav2-velocity-smoother_1.3.2-1.bbappend` – `nav2-voxel-grid_1.3.2-1.bbappend` – `nav2-waypoint-follower_1.3.2-1.bbappend`

Recipe	Description
recipes-qrb-ros	Consists of qrb-ros recipes: – `dmabuf-transport_0.0.0.1.bb` – `lib-mem-dmabuf_0.0.0.1.bb` – `qrb-ros-color-space-convert_0.0.0.1.bb` – `qrb-colorspace-convert-lib_0.0.0.1.bb` – `qrb-sensor-client_0.0.0.1.bb` – `qrb-ros-system-monitor_0.0.0.1.bb` – `qrb-ros-system-monitor-interfaces_0.0.0.1.bb` – `qrb-ros-transport-image-type_0.0.0.1.bb` – `qrb-ros-transport-imu-type_0.0.0.1.bb` – `qrb-ros-transport-point-cloud2-type_0.0.0.1.bb`

meta-qcom-robotics-distro

- **Distro configuration**

`layer.conf`	Configures the project layers with the following information: – Recipe file path – Supported Yocto version
`qcom-robotics-ros2-jazzy.conf`	Configures the distro feature with the following information: – ROS2 jazzy distro enablement – Includes distro conf file of Qualcomm Linux

- **Image recipe**

Recipe	Description
`qcom-robotics-full-image`	Consists of these packages: – ros-core – qirp-sdk – packagegroup-qcom-robotics

- **Package groups**

Package group	Description
packagegroup-qcom-robotics	Package group for upstream basic ROS2 packages, and packages needed for robotics: – ament-cmake – ament-cmake-auto – ament-cmake-core – ament-cmake-export-definitions – ament-cmake-export-dependencies – ament-cmake-export-include-directories – ament-cmake-export-interfaces – ament-cmake-export-libraries – ament-cmake-export-link-flags – ament-cmake-export-targets – ament-cmake-gen-version-h – ament-cmake-gmock – ament-cmake-google-benchmark – ament-cmake-gtest – ament-cmake-include-directories – ament-cmake-libraries – ament-cmake-nose – ament-cmake-pytest – ament-cmake-python – ament-cmake-ros – ament-cmake-target-dependencies – ament-cmake-test – ament-cmake-version – ament-lint-auto – foonathan-memory-staticdev – opencv-staticdev – dmabuf-transport – image-transport – yaml-cpp – camera-info-manager – rclcpp – sensor-msgs – nav-msgs – std-msgs – geometry-msgs – tf2 – tf2-ros – tf2-geometry-msgs – cv-bridge – rosidl-adapter – ncnn-dev – rclcpp-components – rcutils – libgpiod – libgpiod-dev

meta-qcom-robotics-extras

Recipes

Recipe	Description
recipes	**The recipes in `meta-qcom-robotics-extras` override those in `meta-qcom-robotics`. `meta-qcom-robotics` uses the prebuilt binaries by default, while `meta-qcom-robotics-extras` builds the source from Qualcomm proprietary repositories.** Consists of robotics feature recipes: • `auto-explore` • `dfs` • `sensor-service` • `vio` • `depth-vslam` • `mono-vslam` • `stereo-vslam` • `voxel-map`

9.2 References

This document references the following documents and online resources.

- Yocto: https://docs.yoctoproject.org/
- ROS: ROS 2 Documentation: Jazzy
- Qualcomm® Linux:
  - Qualcomm Dragonwing™ RB3 Gen 2 Development Kit Quick Start Guide
  - Qualcomm Dragonwing™ IQ-9075 Evaluation Kit Quick Start Guide
  - Qualcomm® IQ-8 Beta Evaluation Kit Quick Start Guide
  - Qualcomm® Linux Build Guide
  - Qualcomm® Linux Yocto Guide
  - Qualcomm® Linux Virtual Machine Setup Guide
  - Qualcomm® Intelligent Multimedia Product (QIMP) SDK Quick Start Guide
  - Qualcomm IM SDK Reference

LEGAL INFORMATION

Your access to and use of this material, along with any documents, software, specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Material"), is subject to your (including the corporation or other legal entity you represent, collectively "You" or "Your") acceptance of the terms and conditions ("Terms of Use") set forth below. If You do not agree to these Terms of Use, you may not use this Material and shall immediately destroy any copy thereof.

1) Legal Notice.

This Material is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("**Qualcomm Technologies**"), its affiliates and/or licensors described in this Material, and shall not be used for any other purposes. If this Material is marked as "**Qualcomm Internal Use Only**", no license is granted to You herein, and You must immediately (a) destroy or return this Material to Qualcomm Technologies, and (b) report Your receipt of this Material to qualcomm.support@qti.qualcomm.com. This Material may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval, nor may it be used for any machine learning or artificial intelligence development purpose which results, whether directly or indirectly, in the creation or development of an automated device, program, tool, algorithm, process, methodology, product and/or other output. Unauthorized use or disclosure of this Material or the information contained herein is strictly prohibited, and You agree to indemnify Qualcomm Technologies, its affiliates and licensors for any damages or losses suffered by Qualcomm Technologies, its affiliates and/or licensors for any such unauthorized uses or disclosures of this Material, in whole or part.

Qualcomm Technologies, its affiliates and/or licensors retain all rights and ownership in and to this Material. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Material or any information disclosed herein, including, but not limited to, any license to make, use, import or sell any product, service or technology offering embodying any of the information in this Material.

THIS MATERIAL IS BEING PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, WHETHER EXPRESSED, IMPLIED, STATUTORY OR OTHERWISE. TO THE MAXIMUM EXTENT PERMITTED BY LAW, QUALCOMM TECHNOLOGIES, ITS AFFILIATES AND/OR LICENSORS SPECIFICALLY DISCLAIM ALL WARRANTIES OF TITLE, MERCHANTABILITY, NON-INFRINGEMENT, FITNESS FOR A PARTICULAR PURPOSE, SATISFACTORY QUALITY, COMPLETENESS OR ACCURACY, AND ALL WARRANTIES ARISING OUT OF TRADE USAGE OR OUT OF A COURSE OF DEALING OR COURSE OF PERFORMANCE. MOREOVER, NEITHER QUALCOMM TECHNOLOGIES, NOR ANY OF ITS AFFILIATES AND/OR LICENSORS, SHALL BE LIABLE TO YOU OR ANY THIRD PARTY FOR ANY EXPENSES, LOSSES, USE, OR ACTIONS HOWSOEVER INCURRED OR UNDERTAKEN BY YOU IN RELIANCE ON THIS MATERIAL.

Certain product kits, tools and other items referenced in this Material may require You to accept additional terms and conditions before accessing or using those items.

Technical data specified in this Material may be subject to U.S. and other applicable export control laws. Transmission contrary to U.S. and any other applicable law is strictly prohibited.

Nothing in this Material is an offer to sell any of the components or devices referenced herein.

This Material is subject to change without further notification.

In the event of a conflict between these Terms of Use and the *Website Terms of Use* on www.qualcomm.com, the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, or other legal statements or notices found on prior pages of the Material, these Terms of Use will control. In the event of a conflict between these Terms of Use and any other agreement (written or click-through, including, without limitation any non-disclosure agreement) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate and/or licensor with respect to Your access to and use of this Material, the other agreement will control.

These Terms of Use shall be governed by and construed and enforced in accordance with the laws of the State of California, excluding the U.N. Convention on International Sale of Goods, without regard to conflict of laws principles. Any dispute, claim or controversy arising out of or relating to these Terms of Use, or the breach or validity hereof, shall be adjudicated only by a court of competent jurisdiction in the county of San Diego, State of California, and You hereby consent to the personal jurisdiction of such courts for that purpose.

2) Trademark and Product Attribution Statements.

Qualcomm is a trademark or registered trademark of Qualcomm Incorporated. Arm is a registered trademark of Arm Limited (or its subsidiaries) in the U.S. and/or elsewhere. The Bluetooth® word mark is a registered trademark owned by Bluetooth SIG, Inc. Other product and brand names referenced in this Material may be trademarks or registered trademarks of their respective owners.

Snapdragon and Qualcomm branded products referenced in this Material are products of Qualcomm Technologies, Inc. and/or its subsidiaries. Qualcomm patented technologies are licensed by Qualcomm Incorporated.