Qualcomm

Qualcomm Technologies, Inc.

Qualcomm Linux Build Guide

80-70014-254 Rev. AN

August 7, 2024

Contents

Tables

1 Introduction

This guide describes the methods to configure, download, compile, and flash Qualcomm® Linux® and the associated firmware on supported devices. **This information is also available in** Simplified Chinese.

Qualcomm recommends that you read the Qualcomm Linux Yocto Guide before starting your build.

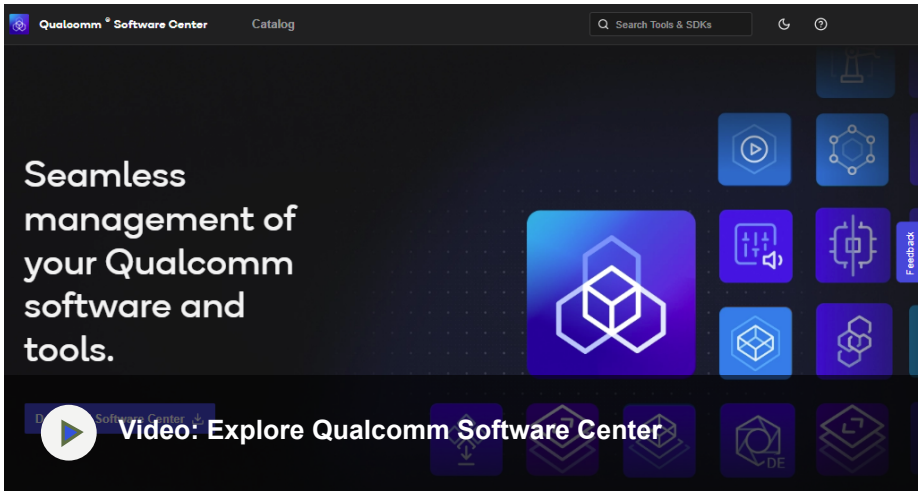
Unregistered users

Unregistered users can sync and build Qualcomm Linux using the GitHub workflow for unregistered users.

Registered users

Registered users can use any one of the following three methods to sync and build Qualcomm Linux. These methods use the Qualcomm Yocto layers and the supporting base Yocto layers.

Launcher	CLI	GitHub
Easy-to-use, GUI-based, Qualcomm Software Center (QSC) Launcher.	Simple QSC command-line interface (CLI).	Instructions to use GitHub based workflow.
Build with QSC Launcher	Build with QSC CLI	GitHub workflow for registered users



NOTE Prebuilt binaries along with Platform eSDK links are hosted in the Release Notes.

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

applications and open-source plugins. For more details, see How to download the Platform eSDK?.

NOTE Qualcomm Linux allows you to build images for QCS6490 and QCS5430.

QCM6490 and QCS6490 are used interchangeably in this guide.

2 Build with QSC Launcher

The following sections provide instructions to build Qualcomm Linux using the QSC Launcher.

2.1 Host machine requirements

Configuration	Tools	Permissions
x86 machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to execute a few commands.
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	Tar 1.28 or later versions	
300 GB free disk space (swap partition > 32 GB)	Python 3.10.2 or later versions	
16 GB RAM	GCC 7.5 or later versions	
Ubuntu 22.04	GNU Make 4.0 or later versions	

2.2 Install QSC

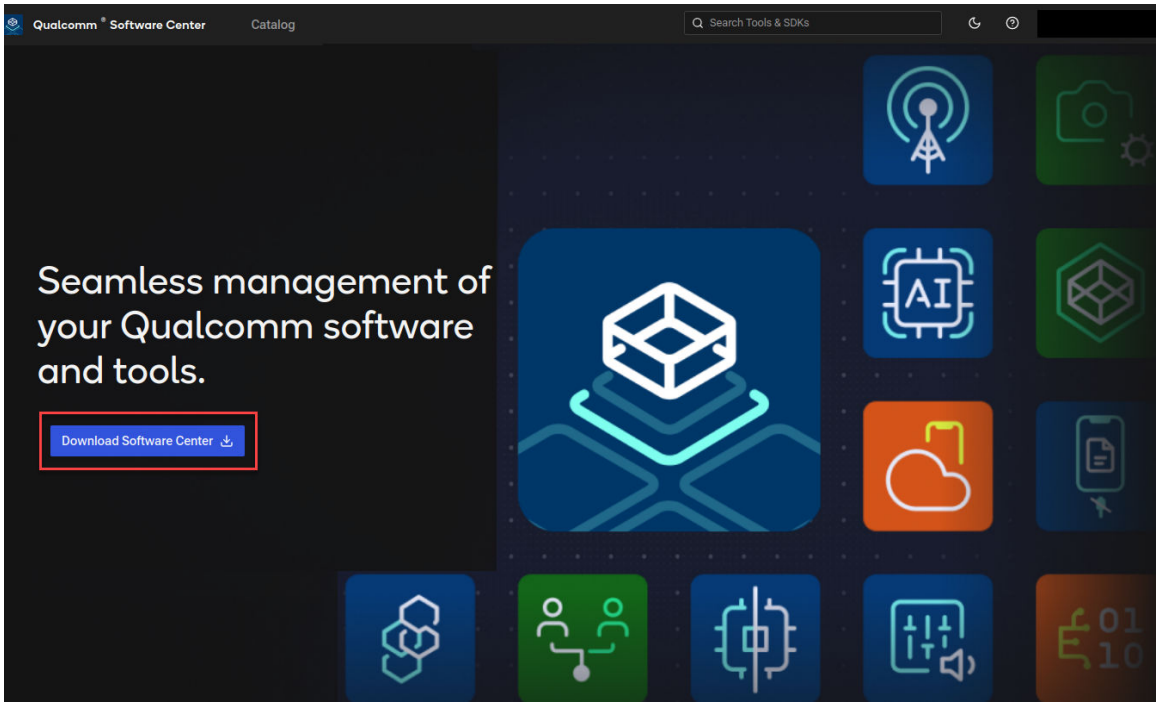
There are two methods to install QSC:

- Using a GUI
- Using a CLI

Install QSC using a GUI

Use the Qualcomm Software Center (QSC) GUI to install QSC.

1. Use your Qualcomm ID to log in tohttps://softwarecenter.qualcomm.com, then click **Download Software Center**. The QSC Debian package (`.deb`) downloads to your machine.



2. Install the downloaded QSC Debian package:

```
sudo dpkg -i <download_path>/QualcommSoftwareCenterx.x.x.linux-x86.deb
# <x.x.x> indicates QSC version
```

Install QSC using a CLI

1. Install curl (if not already installed):

```
sudo apt install curl
```

2. Download the QSC Debian package:

```
cd <workspace_path>
curl -L https://softwarecenter.qualcomm.com/api/download/software/qsc/
linux/latest.deb -o qsc_installer.deb
```

3. Install the Debian package:

```
sudo dpkg -i qsc_installer.deb
```

After a successful installation, the message `Installed Qualcomm Software Center vx.x.x successfully` is displayed.

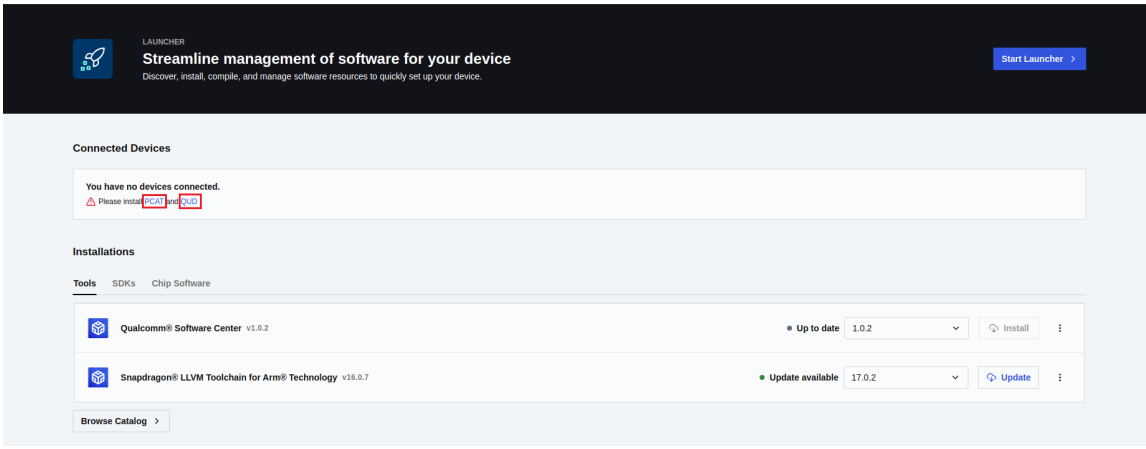
2.3 Use QSC Launcher

NOTE A one-time login is required into chipcode.qti.qualcomm.com to download Qualcomm proprietary git repositories. Use your Qualcomm login credentials to complete this step.

1. To open the QSC desktop application, either launch **Qualcomm Software Center** from the **Applications** menu or run the following command from the Linux terminal:

   ```
   /opt/qcom/softwarecenter/bin/softwarecenter
   ```

   NOTE For the Launcher workflow to detect connected devices and flash software builds, ensure that the Qualcomm Product Configuration Assistant Tool (PCAT) and Qualcomm USB Driver (QUD) are installed on the host machine. Click **PCAT** to install PCAT and **QUD** to install QUD as shown in the following image:



   Or

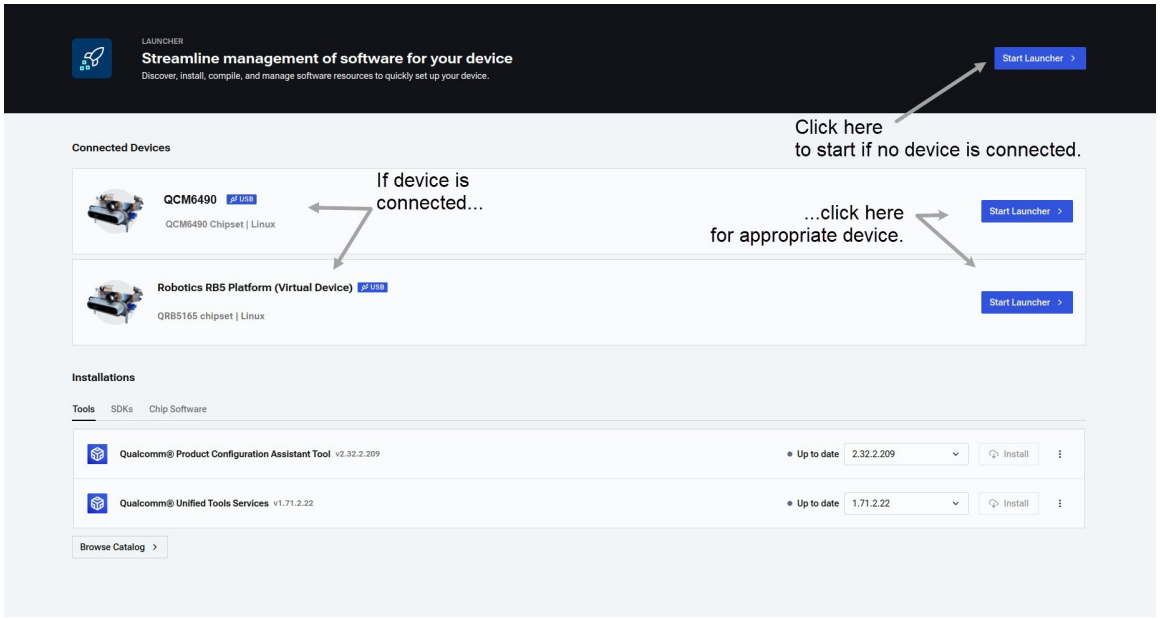
   Install PCAT and QUD using `qpm-cli`:

   ```
   qpm-cli --login
   qpm-cli --install pcat --activate-default-license
   qpm-cli --install qud --activate-default-license
   ```

   The `qpm-cli --help` command lists the help options.

   For Ubuntu 22.04, you may encounter an issue while installing QUD where you are asked to enrol the public key on your Linux host for a successful QUD installation. For more details, follow the steps provided in the `signReadme.txt` file available at the `/opt/QUIC/sign/` directory.

2. Use your Qualcomm ID to log in to the QSC desktop application. A dashboard page appears as shown in the following figure:

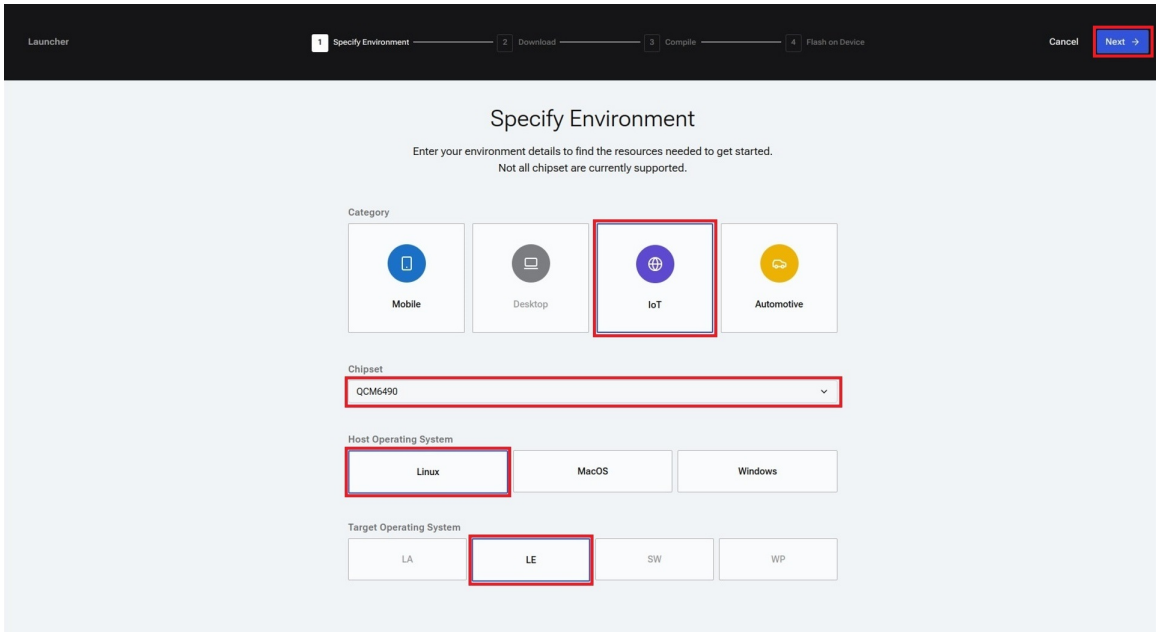


   - If you do not have a connected device, click **Start Launcher** on the top panel to start the steps to configure, download, compile, and flash Qualcomm Linux to your device.
   - If you have a connected device, click **Start Launcher** for the appropriate device in the **Connected devices** panel to download, compile, and flash Qualcomm Linux to your connected device.

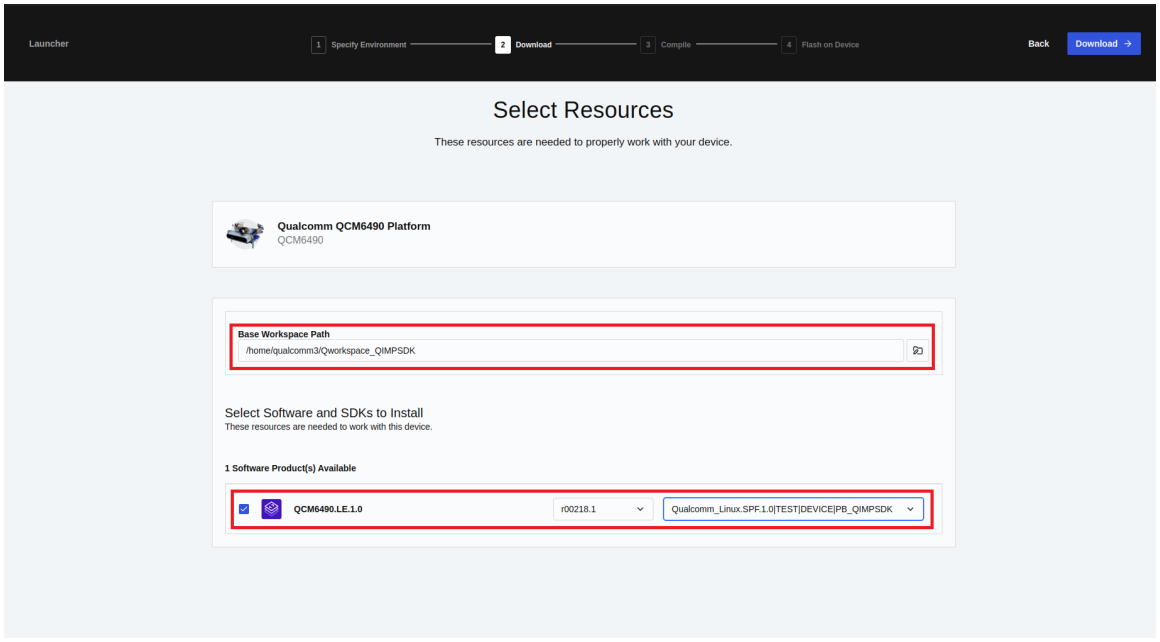
3. On the **Specify Environment** page, select the following values based on the build:
   - Category: **IoT**
   - Chipset: **QCM6490**

- Host OS: **Linux**
- Target OS: **LE**

NOTE See Release Notes for all the supported chipsets.



4. Click **Next**. The **Select Resources** page appears.
5. On the **Select Resources** page, perform the following steps:



a. In the **Base Workspace Path** text box, specify a directory path where you want to download the software. You can click the select icon to display the directory selection window.

b. Select product ID (For chipset QCM6490 and target OS LE, **QCM6490.LE.1.0** is the product ID).

c. Select release ID (See the latest Release Notes. For example, **r00218.1**).

d. Select the appropriate distribution to download. Distribution access is controlled by access levels as listed in the following table:

NOTE For more details on the available distributions, see the **Access Controlled Distribution** table in the Release Notes.

Access level	Distribution	Yocto layers
Public developer (unregistered)	Base build: High-level operating system (OS) and prebuilt firmware (GPS only) `Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PUBLIC`	`meta-qcom` `meta-qcom-hwe`

	Base build + Qualcomm Intelligent Multimedia Product (QIMP) SDK
	`Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-qim-product-sdk`

	Base build + QIMP SDK + Qualcomm Intelligent Robotics Product (QIRP) SDK
	`Qualcomm_Linux.SPF.1.0|TEST|DEVICE|RoboApiLnx`
		`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	Base build: High-level OS and firmware source (GPS only)
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

	Base build + QIMP SDK
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

	Base build + QIMP SDK + QIRP SDK
	`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`

Access level	Distribution	Yocto layers

Licensed developer with additional access	Base build: High-level OS and firmware (GPS only) source
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

	Base build + QIMP SDK (GPS only)
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

	Base build: High-level OS and firmware (GPS and modem) source
	`Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

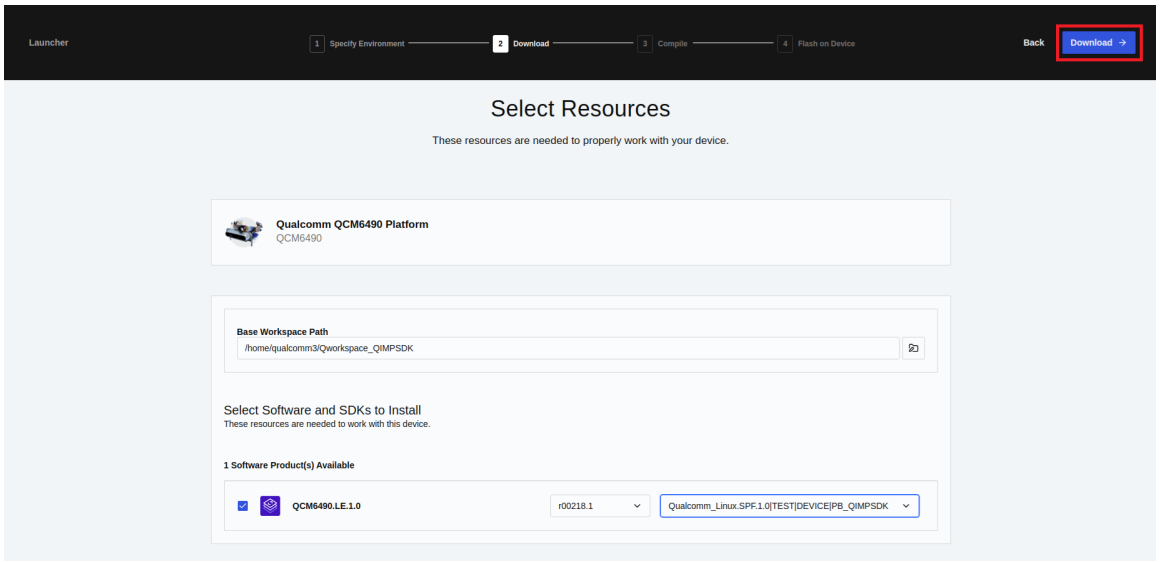
	Base build + QIMP SDK (GPS and modem)
	`Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

The Qualcomm Linux Yocto layers are described in the following table:

Table 2-1 Qualcomm Linux Yocto layers

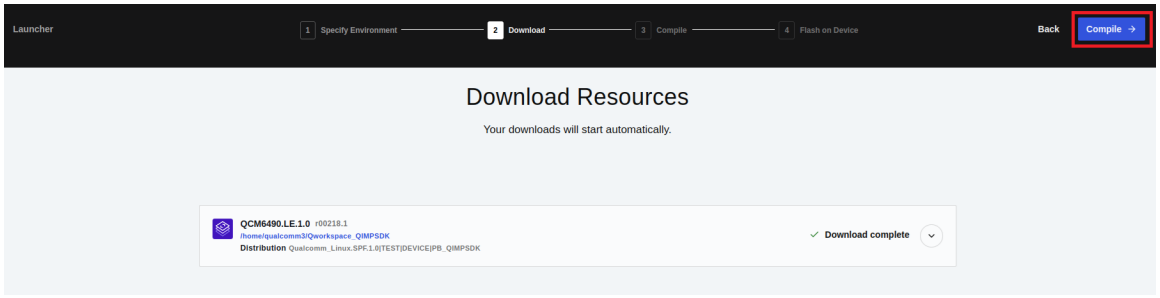
Yocto layer	Description
`meta-qcom`	Contains Qualcomm hardware support metadata with upstream OSS software components.
`meta-qcom-hwe`	Contains Qualcomm hardware support metadata with board support package (BSP) software components.
`meta-qcom-extras`	Enables source compilation of select components, which are otherwise present as binary in `meta-qcom-hwe`. This layer is an optional metadata layer for registered users.
`meta-qcom-qim-product-sdk`	Provides Qualcomm's multimedia and AI SDKs based GStreamer framework. It includes a set of GStreamer plugin sample applications for multimedia and AI use cases.
`meta-ros`	Contains a series of OpenEmbedded layers to add support for the Robot Operating System (ROS) for embedded Linux releases by the Yocto Project.
`meta-qcom-robotics-distro`	Contains the configuration information needed to generate the ROS image, including the package group and image recipe.
`meta-qcom-robotics`	Contains the robotics recipes and the mechanism to generate Robotics SDK.
`meta-qcom-robotics-extras`	Contains the proprietary robotics recipes that are built with source.
`meta-qcom-robotics-sdk`	Contains the generation mechanism (package of cross-compile toolchain, script, and function SDK) and pick-up mechanism (necessary files from QIMP SDK, QNN SDK, and QIRF SDK through `config.json`) of Robotics Product SDK.

6. Click **Download** to download the selected software:

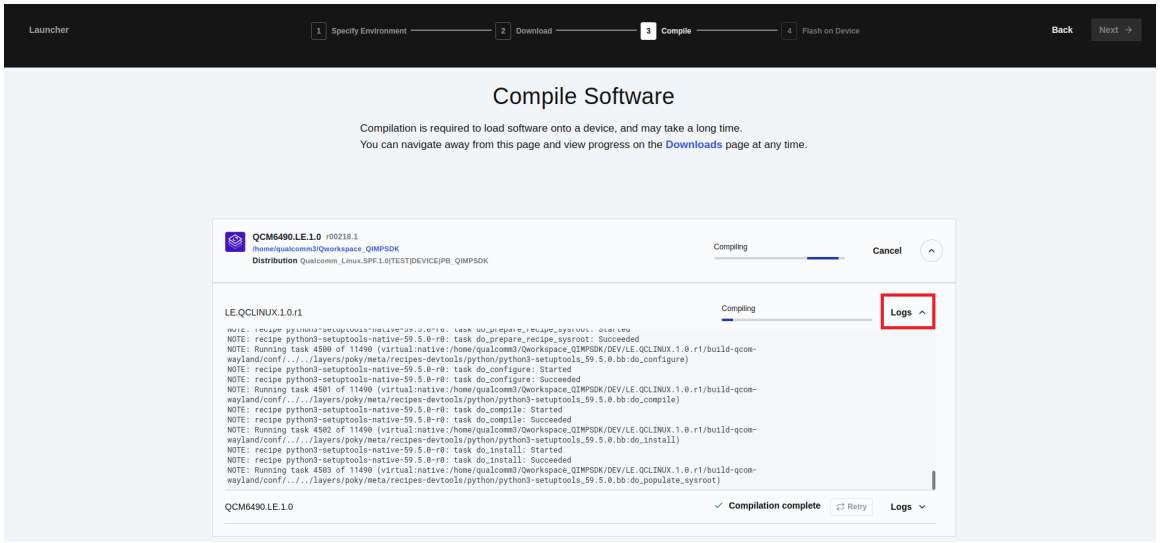


The **Download** page displays the download progress. Download progress is also available in the top menu bar **Downloads** option in the Chip Software section.

7. To start compilation after the download completes, select **Compile** (depending on the size of the downloaded software and host machine configuration, compilation may take a few hours):



8. To view the compilation progress of individual software images, expand the logs panel as shown in the following figure:



On a successful build of the `qcom-wayland` distributions, you can see the images at the following path:

```
# system.img is present at the following path
<Base_Workspace_Path>/DEV/LE.QCLINUX.1.0.r1/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image/*
```

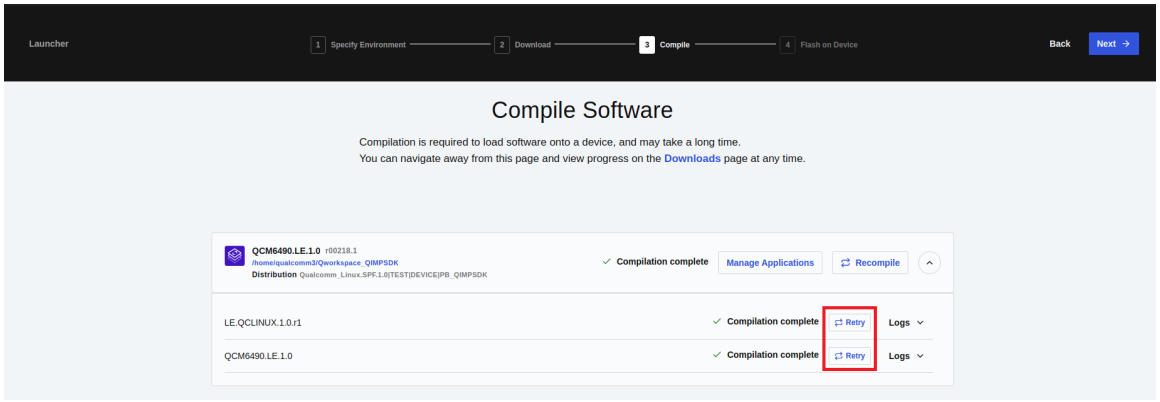
On a successful build of the `qcom-robotics-ros2-humble` (QIRP) distribution, you can see the QIRP SDK build artifacts at the following paths:

```
QIRP SDK artifacts: <Base_Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcom-robotics-ros2-humble/tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk_<version>.tar.gz
# system.img is present at the following path
Robotics image: <Base_Workspace_Path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcom-robotics-ros2-humble/tmp-glibc/deploy/images/qcm6490/qcom-robotics-full-image/*
```

NOTE `<Base_Workspace_Path>` is the path that you select on the Select Resources page.

NOTE BitBake fetch errors are typically intermittent fetch failures. Retry step 7 to work around these intermittent errors. If the issue persists, see BitBake Fetcher Error for a solution.

9. To incorporate changes made after the compilation step is complete, click **Retry** to recompile:



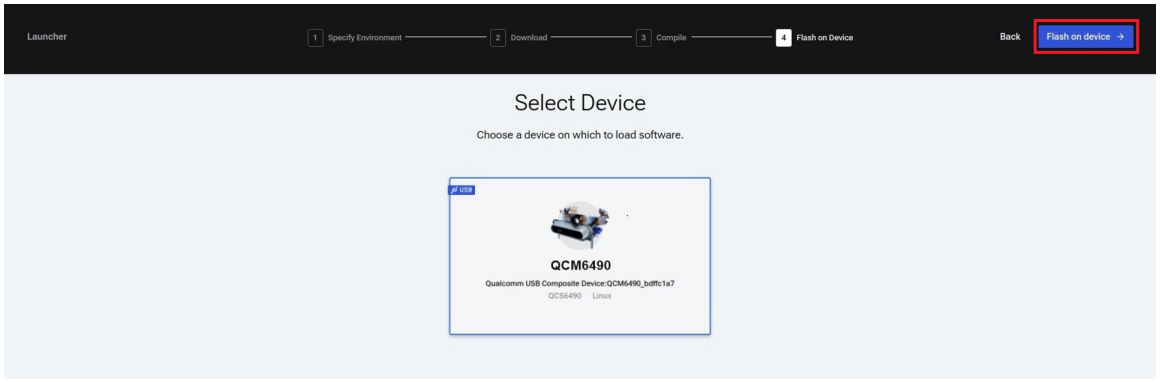
10. Click **Next** to flash software to the device.

Flash

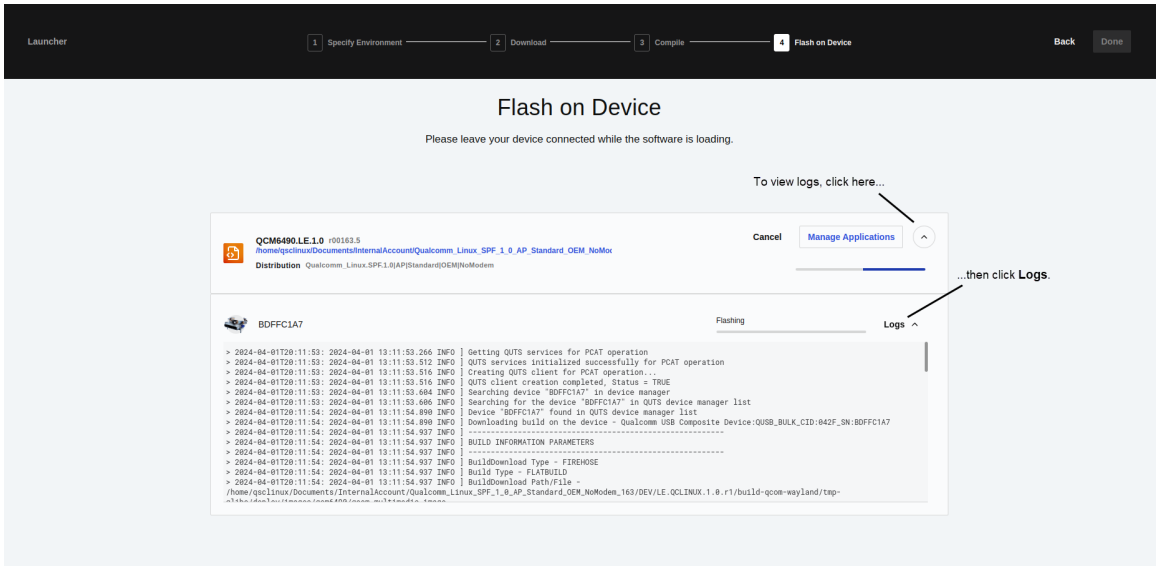
NOTE Ensure that the device is in Emergency Download (EDL) mode before you flash the software. For more information on how to force the device into EDL mode, see Move to EDL mode.

To flash software to the device with Launcher, perform the following steps:

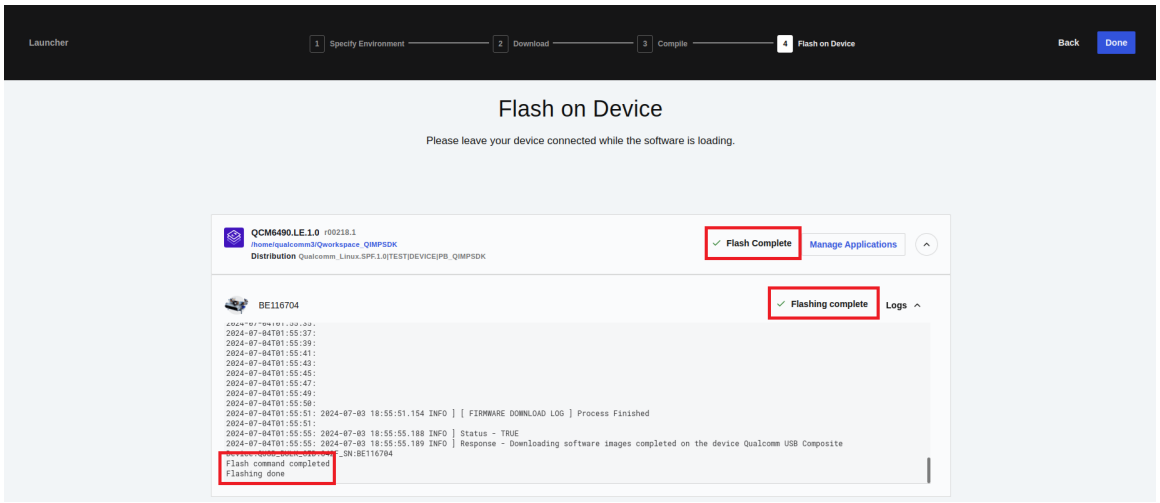
1. With the device connected and in EDL mode, select the device on which you want to flash the software as shown in the following figure:



2. Click **Flash on device**. The page updates and displays a progress bar as Launcher begins flashing the software. Leave the device connected while the software is being flashed.
3. To view logs, expand the logs panel as shown in the following figure:



4. When the process is finished, *Flash Complete* and similar messages are displayed on the page:



5. Click **Done**. To connect to the device, see How to SSH.

3 Build with QSC CLI

The following sections provide instructions to build Qualcomm Linux using the QSC CLI.

3.1 Host machine requirements

Configuration	Tools	Permissions
x86 machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to execute a few commands.
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	Tar 1.28 or later versions	
300 GB free disk space (swap partition > 32 GB)	Python 3.10.2 or later versions	
16 GB RAM	GCC 7.5 or later versions	
Ubuntu 22.04	GNU Make 4.0 or later versions	

3.2 Install QSC CLI

Set up `qsc-cli`:

1. Install curl (if not already installed):

   ```
   sudo apt install curl
   ```

2. Download the Debian package for `qsc-cli`:

   ```
   cd <workspace_path>
   curl -L https://softwarecenter.qualcomm.com/api/download/software/qsc/
   linux/latest.deb -o qsc_installer.deb
   ```

3. Install the `qsc-cli` Debian package:

   ```
   sudo dpkg -i qsc_installer.deb
   ```

4. Log in to `qsc-cli`:

   ```
   qsc-cli login -u <username>
   ```

   NOTE A one-time login into chipcode.qti.qualcomm.com is required to download Qualcomm proprietary git repositories. Use your Qualcomm login credentials to complete this step.

   For more details, see `qsc-cli` related topics in How to.

3.3 Use QSC CLI

This section provides instructions on how to download, compile, and recompile Qualcomm Linux using QSC CLI.

Download

NOTE If you are building a distribution with access level "Registered developer from a verified organization" or "Licensed customers with additional access", then you must log in to `qpm-cli` before you compile:

```
# Run the following command to login
qpm-cli --login
# Run the following command to verify if qpm-cli login is successful
qpm-cli --product-list
```

This command prompts for your username and password. After successfully logging in, the system fetches and refreshes the product list.

- Download a particular software release by specifying the product ID, build ID, distribution, and the absolute/full workspace path as shown in the following example:

```
# qsc-cli download --workspace-path '<absolute_workspace_path>' --product
'<Product_ID>' --build '<Build_ID>' --distribution '<Distro>'
```

  - Select Product_ID and Build_ID values from the **QSC-CLI Input Parameters** table in the Release Notes.

    Example: For QCM6490 with Linux embedded (LE) build, the input parameters are shown in the following table:

Product_ID (--product)	`QCM6490.LE.1.0`
Build_ID (--build)	`QCM6490.LE.1.0-00218-STD.PROD-1`

  - Select the appropriate distribution to download. Distribution access is controlled by access levels as listed in the following table:

    NOTE For more details on the available distributions, see **Access Controlled Distribution** table in the Release Notes.

Access level	Distribution	Yocto layers
Unregistered/community developer	Base build: High-level operating system (OS) and prebuilt firmware (GPS only) `Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PUBLIC`	`meta-qcom` `meta-qcom-hwe`

	Base build + Qualcomm Intelligent Multimedia Product (QIMP) SDK
	`Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-qim-product-sdk`

Access level	Distribution	Yocto layers

	Base build + QIMP SDK + Qualcomm Intelligent Robotics Product (QIRP) SDK
	`Qualcomm_Linux.SPF.1.0|TEST|DEVICE|RoboApiLnx`
		`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	Base build: High-level OS and firmware source (GPS only)
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

	Base build + QIMP SDK
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

	Base build + QIMP SDK + QIRP SDK
	`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`

Licensed developer with additional access	Base build: High-level OS and firmware (GPS only) source
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

	Base build + QIMP SDK (GPS only)
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

Access level	Distribution	Yocto layers

	Base build: High-level OS and firmware (GPS and modem) source
	`Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

	Base build + QIMP SDK (GPS and modem)
	`Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

For Yocto layer descriptions, see Table 2-1.

- Start the download:

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
# WORKSPACE_DIR=/local/mnt/workspace/Qworkspace_QIMPSDK

# Example
qsc-cli download --workspace-path '/local/mnt/workspace/
Qworkspace_QIMPSDK' --product 'QCM6490.LE.1.0' --build
'QCM6490.LE.1.0-00218-STD.PROD-1' --distribution 'Qualcomm_Linux.SPF.1.0|
TEST|DEVICE|PB_QIMPSDK'
```

When the process is completed successfully, the software product is available in the user-provided workspace directory.

NOTE A new workspace is required for each distribution if you are downloading more than one distribution.

Compile

Start the compilation after the download completes:

NOTE Depending on the size of the software and host machine configuration, compilation may take a few hours.

```
qsc-cli compile --workspace-path '<absolute_workspace_path>'

# Example
qsc-cli compile --workspace-path '/local/mnt/workspace/Qworkspace_QIMPSDK'
```

This process builds the Qualcomm firmware as needed and also completes the build for the Qualcomm Linux.

NOTE If you see a BitBake fetcher error, retry compilation to work around this error. If the issue persists, see BitBake Fetcher Error for a solution.

On a successful build of the `qcom-wayland` distributions, you can see the images at the following path:

```
# system.img is present at the following path
<workspace_path>/DEV/LE.QCLINUX.1.0.r1/build-qcom-wayland/tmp-glibc/
deploy/images/qcm6490/qcom-multimedia-image/*
```

On a successful build of the `qcom-robotics-ros2-humble` (QIRP) distribution, you can see the QIRP SDK build artifacts at the following paths:

```
QIRP SDK artifacts: <workspace_path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcom-
robotics-ros2-humble/tmp-glibc/deploy/qirpsdk_artifacts/qirp-
sdk_<version>.tar.gz
# system.img is present at the following path
Robotics image: <workspace_path>/DEV/LE.QCROBOTICS.1.0.r1/build-qcom-
robotics-ros2-humble/tmp-glibc/deploy/images/qcm6490/qcom-robotics-full-
image/*
```

Recompile

Recompile your workspace if you already have a workspace built using QSC CLI:

```
# qsc-cli compile --image '<software_image_name>' --workspace-path
'<absolute_workspace_path>'

# Example
qsc-cli compile --image LE.QCLINUX.1.0.r1 --workspace-path '/local/mnt/
workspace/Qworkspace_QIMPSDK'
```

NOTE For more information on software image names (`--image`), see QSC-CLI input parameters table in the Release Notes.

Flash

For the steps to flash software images to the device, see Flash images for registered users.

4 GitHub workflow for unregistered users

The following sections provide instructions to use GitHub to make a build utilizing the prebuilt proprietary binary images.

4.1 Host machine requirements

Configuration			Tools	Permissions
Linux	**Windows**	**Mac**		
x86 machine	x86 machine	x86/Arm® machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to execute a few commands.
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	16-core CPU	16-core CPU	Git 1.8.3.1 or later versions	
300 GB free disk space (swap partition > 32 GB)	300 GB free space for the VirtualBox VM	300 GB free space for UTM	Python 3.10.2 or later versions	
16 GB RAM	16 GB RAM	16 GB RAM	GCC 7.5 or later versions	
Ubuntu 22.04	Microsoft Windows 11 OS	Apple® Mac® OS 14	GNU Make 4.0 or later versions	

NOTE To set up a virtual machine (VM) running Ubuntu 22.04 OS on Microsoft Windows or Apple Mac, see Qualcomm Linux Virtual Machine Setup Guide. Code compilation on a VM is a slow process and may take a few hours. Qualcomm recommends using an Ubuntu host machine for compilation.

4.2 Build with standalone commands

Ubuntu host setup

The Ubuntu host machine needs a few setup operations to ensure that the required software tools are installed and configured for use.

1. Install the following packages to prepare your host environment for Yocto build:

```
sudo apt update
sudo apt install repo gawk wget git diffstat unzip texinfo gcc build-
essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils
```

```
debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-
dev pylint xterm python3-subunit mesa-common-dev zstd liblz4-tool locales
tar python-is-python3 file libxml-opml-simplegen-perl vim whiptail
```

2. Set up the locales (if not set up already):

```
sudo locale-gen en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

3. Update git configurations:

```
# Check if your identity is configured in .gitconfig
git config --get user.email
git config --get user.name

# Run the following commands if you do not have your account identity set
in .gitconfig
git config --global user.email <Your email ID>
git config --global user.name <"Your Name">

# Add the following UI color option for output of console (optional)
git config --global color.ui auto

# Add the following git configurations to fetch large size repositories
and to avoid unreliable connections
git config --global http.postBuffer 1048576000
git config --global http.maxRequestBuffer 1048576000
git config --global http.lowSpeedLimit 0
git config --global http.lowSpeedTime 999999
```

Sync

This section uses the Repo tool installed in the previous section to download a list of git repositories and additional attributes from the Qualcomm manifest. As part of this process, it downloads the manifest used in the `repo init` command.

The following table shows an example mapping of Yocto layers to the manifest release tags that are used to download and build Qualcomm Linux:

Yocto layers	Manifest release tag	Distribution (DISTRO)
▪ meta-qcom ▪ meta-qcom-hwe	Base build: High-level OS and prebuilt firmware (GPS only) qcom-6.6.28-QLI.1.1-Ver.1.1.xml	qcom-wayland
▪ meta-qcom ▪ meta-qcom-hwe ▪ meta-qcom-qim-product-sdk	Base build + QIMP SDK build: qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3.xml	qcom-wayland

Yocto layers	Manifest release tag	Distribution (DISTRO)
▪ meta-qcom ▪ meta-qcom-hwe ▪ meta-qcom-realtime	Base build + Real-time kernel build: qcom-6.6.28-QLI.1.1-Ver.1.1_realtime-linux-1.0.xml	qcom-wayland
▪ meta-qcom ▪ meta-qcom-hwe ▪ meta-ros ▪ meta-qcom-robotics ▪ meta-qcom-robotics-distro ▪ meta-qcom-robotics-sdk ▪ meta-qcom-qim-product-sdk	Base build + QIRP SDK build: qcom-6.6.28-QLI.1.1-Ver.1.1_robotics-product-sdk-1.1.xml	qcom-robotics-ros2-humble

For Yocto layer descriptions, see Table 2-1.

Build base image

1. Download Qualcomm Yocto and supporting layers:

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
```

   NOTE For the latest <manifest release tag>, see the *Build-critical release tags* section in the Release Notes.

2. Set up the build environment:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

3. Build the software image:

   NOTE For supported image recipes, see What are the image recipes supported as part of the GitHub workflow?.

```
bitbake <image recipes>
bitbake qcom-multimedia-image
```

On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
qcom-multimedia-image
ls -al system.img
```

Build QIMP SDK image

1. Download Qualcomm Yocto and supporting layers:

   NOTE The `<manifest release tag>` for the QIMP SDK build is the same as the base build. The QIMP SDK layer must be cloned on top of the base build.

   For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
```

2. Clone QIMP SDK layer into the workspace:

```
git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk -b <qim-
product-sdk release tag> layers/meta-qcom-qim-product-sdk
# Example, <qim-product-sdk release tag> is qcom-6.6.28-QLI.1.1-
Ver.1.1_qim-product-sdk-1.1.3
```

   To build a QIMP SDK layer, the following export is required:

```
export EXTRALAYERS="meta-qcom-qim-product-sdk"
```

3. Set up the build environment:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

4. Build the software image:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qim-product-sdk
```

On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image
ls -al system.img
```

Build QIRP SDK image

1. Download Qualcomm Yocto and supporting layers:

   NOTE The `<manifest release tag>` for QIRP SDK build is the same as the base build. QIRP SDK layers must be cloned on top of the base build.

   ```
   # cd to directory where you have 300 GB of free storage space to create your workspaces
   mkdir <WORKSPACE_DIR>
   cd <WORKSPACE_DIR>
   repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-kirkstone -m <manifest release tag>
   # Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
   repo sync
   ```

   NOTE For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

2. Download the QIRP SDK layers into the base build `<WORKSPACE DIR>` directory:

   ```
   git clone https://git.codelinaro.org/clo/le/meta-ros.git -b ros.qclinux.1.0.r1-rel layers/meta-ros
   git clone https://github.com/quic-yocto/meta-qcom-robotics.git layers/meta-qcom-robotics
   git clone https://github.com/quic-yocto/meta-qcom-robotics-distro.git layers/meta-qcom-robotics-distro
   git clone https://github.com/quic-yocto/meta-qcom-robotics-sdk.git layers/meta-qcom-robotics-sdk
   git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk layers/meta-qcom-qim-product-sdk
   ```

3. Set up the build environment:

   ```
   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=qcm6490 DISTRO=qcom-robotics-ros2-humble source setup-robotics-environment
   # source setup-robotics-environment: Sets the environment settings, creates the build directory build-qcom-robotics-ros2-humble,
   # and enters into build-qcom-robotics-ros2-humble directory.
   ```

4. Build the robotics image and QIRP SDK artifacts:

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

On a successful build, you can see the QIRP SDK build artifacts at the following paths:

```
QIRP SDK artifacts: <WORKSPACE DIR>/build-qcom-robotics-ros2-humble/
tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk_<version>.tar.gz
# system.img is present in the following path
Robotics image: <WORKSPACE DIR>/build-qcom-robotics-ros2-humble/tmp-
glibc/deploy/images/qcm6490/qcom-robotics-full-image
```

Build real-time Linux image

1. Download Qualcomm Yocto and supporting layers:

   NOTE The `<manifest release tag>` for real-time Linux image is the same as the base build. Real-time Linux must be cloned on top of the base build.

   ```
   # cd to directory where you have 300 GB of free storage space to create
   your workspaces
   mkdir <WORKSPACE_DIR>
   cd <WORKSPACE_DIR>
   repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
   kirkstone -m <manifest release tag>
   # Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
   repo sync
   ```

   NOTE For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

2. Clone real-time Linux layer into the workspace:

   ```
   git clone https://github.com/quic-yocto/meta-qcom-realtime -b <meta-qcom-
   realtime release tag> layers/meta-qcom-realtime
   # Example, <meta-qcom-realtime release tag> is qcom-6.6.28-QLI.1.1-
   Ver.1.1_realtime-linux-1.0
   ```

   To build a real-time layer, the following export is required:

   ```
   export EXTRALAYERS="meta-qcom-realtime"
   ```

3. Set up the build environment:

   ```
   MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
   # source setup-environment: Sets the environment settings, creates the
   build directory build-qcom-wayland,
   # and enters into build-qcom-wayland directory
   ```

4. Build the software image:

   ```
   bitbake qcom-multimedia-image
   ```

   On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

   ```
   cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
   qcom-multimedia-image
   ls -al system.img
   ```

Flash

Flash software images to the device using Flash images for unregistered users.

5 GitHub workflow for registered users

The following sections provide instructions to use GitHub to make a build utilizing prebuilt proprietary binaries/images.

5.1 Host machine requirements

Configuration			Tools	Permissions
Linux	**Windows**	**Mac**		
x86 machine	x86 machine	x86/Arm® machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to execute a few commands.
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	16-core CPU	16-core CPU	Git 1.8.3.1 or later versions	
300 GB free disk space (swap partition > 32 GB)	300 GB free space for the VirtualBox VM	300 GB free space for UTM	Python 3.10.2 or later versions	
16 GB RAM	16 GB RAM	16 GB RAM	GCC 7.5 or later versions	
Ubuntu 22.04	Microsoft Windows 11 OS	Apple® Mac® OS 14	GNU Make 4.0 or later versions	

NOTE To set up a virtual machine (VM) running Ubuntu 22.04 OS on Microsoft Windows or Apple Mac, see Qualcomm Linux Virtual Machine Setup Guide. Code compilation on a VM is a slow process and may take a few hours. Qualcomm recommends using an Ubuntu host machine for compilation.

5.2 Install QSC CLI

Set up `qsc-cli`:

1. Install curl (if not already installed):

   ```
   sudo apt install curl
   ```

2. Download the Debian package for `qsc-cli`:

   ```
   cd <workspace_path>
   curl -L https://softwarecenter.qualcomm.com/api/download/software/qsc/
   linux/latest.deb -o qsc_installer.deb
   ```

3. Install the `qsc-cli` Debian package:

```
sudo dpkg -i qsc_installer.deb
```

4. Log in to `qsc-cli`:

```
qsc-cli login -u <username>
```

NOTE A one-time login into chipcode.qti.qualcomm.com is required to download Qualcomm proprietary git repositories. Use your Qualcomm login credentials to complete this step.

For more details, see `qsc-cli` related topics in How to.

5.3 Workflow options

There are two workflows for registered users to set up, sync, and build Qualcomm Linux with GitHub as described in the following table:

Standalone Commands	docker
Build public Qualcomm Yocto layers with standalone commands.	Build public Qualcomm Yocto layers with Dockerfile.
Build with standalone commands	Build with Dockerfile

5.4 Build with standalone commands

Ubuntu host setup

The Ubuntu host machine needs a few setup operations to ensure that the required software tools are ready.

1. Install the following packages to prepare your host environment for Yocto build:

```
sudo apt update
sudo apt install repo gawk wget git diffstat unzip texinfo gcc build-
essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils
debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-
dev pylint xterm python3-subunit mesa-common-dev zstd liblz4-tool locales
tar python-is-python3 file libxml-opml-simplegen-perl vim whiptail
```

2. Add your Qualcomm login ID with Personalized Access Token (PAT) to the `.netrc` file in your home directory:

```
# Log in to qsc-cli to generate PAT
qsc-cli login -u <username>
# Run the following command to generate PAT
qsc-cli pat --get
# This command gives output as shown in the following note
# The last line in this output is the token, which can be used to access
# Qualcomm Proprietary repositories. This token expires in two weeks.
```

NOTE user@hostname:/local/mnt/workspace$ qsc-cli pat --get

[Info]: Starting qsc-cli version 0.0.0.9

5LThNIkIb55mMVLB5C2KqUGU2jCF

```
vim ~/.netrc # add the following entries

machine chipmaster2.qti.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>

machine qpm-git.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>
```

3. Set up the locales (if not set up already):

```
sudo locale-gen en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

4. Update git configurations:

```
# Check if your identity is configured in .gitconfig
git config --get user.email
git config --get user.name

# Run the following commands if you do not have your account identity set
in .gitconfig
git config --global user.email <Your email ID>
git config --global user.name <"Your Name">

# Add the following UI color option for output of console (optional)
git config --global color.ui auto

# Add the following git configurations to fetch large size repositories
and to avoid unreliable connections
git config --global http.postBuffer 1048576000
git config --global http.maxRequestBuffer 1048576000
git config --global http.lowSpeedLimit 0
git config --global http.lowSpeedTime 999999

# Add the following git configurations to follow remote redirects from
http-alternates files or alternates
git config --global http.https://
chipmaster2.qti.qualcomm.com.followRedirects true
git config --global http.https://qpm-git.qualcomm.com.followRedirects true
```

Sync

This section uses the Repo tool installed in the previous section to download a list of git repositories and additional attributes from the Qualcomm manifest. As part of this process, it downloads the manifests used in the `repo init` command.

The following table shows an example mapping of Yocto layers to the manifest release tags that are used to download and build Qualcomm Linux:

Yocto layers	Manifest release tag	Distribution (DISTRO)
▪ meta-qcom ▪ meta-qcom-hwe	Base build: High-level OS and prebuilt firmware (GPS only) qcom-6.6.28-QLI.1.1-Ver.1.1.xml	qcom-wayland
▪ meta-qcom ▪ meta-qcom-hwe ▪ meta-qcom-qim-product-sdk	Base build + QIMP SDK build: qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3.xml	qcom-wayland
▪ meta-qcom ▪ meta-qcom-hwe ▪ meta-qcom-realtime	Base build + Real-time kernel build: qcom-6.6.28-QLI.1.1-Ver.1.1_realtime-linux-1.0.xml	qcom-wayland
▪ meta-qcom ▪ meta-qcom-hwe ▪ meta-ros ▪ meta-qcom-robotics ▪ meta-qcom-robotics-distro ▪ meta-qcom-robotics-sdk ▪ meta-qcom-qim-product-sdk	Base build + QIRP SDK build: qcom-6.6.28-QLI.1.1-Ver.1.1_robotics-product-sdk-1.1.xml	qcom-robotics-ros2-humble

For Yocto layer descriptions, see Table 2-1.

NOTE For information on building the `meta-qcom-extras` add-on layer and select firmware sources, see GitHub workflow (firmware and extras).

Build base image

1. Download Qualcomm Yocto and supporting layers:

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
```

NOTE For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

2. Set up the build environment:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

3. Build the software image:

   NOTE For supported image recipes, see What are the image recipes supported as part of the GitHub workflow?.

```
bitbake <image recipes>
bitbake qcom-multimedia-image
```

   On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
qcom-multimedia-image
ls -al system.img
```

Build QIMP SDK image

1. Download Qualcomm Yocto and supporting layers:

   NOTE The `<manifest release tag>` for QIMP SDK build is the same as the base build. QIMP SDK layer must be cloned on top of the base build.

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
```

   NOTE For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

2. Clone QIMP SDK layer into the workspace:

```
git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk -b <qim-
product-sdk release tag> layers/meta-qcom-qim-product-sdk
# Example, <qim-product-sdk release tag> is qcom-6.6.28-QLI.1.1-
Ver.1.1_qim-product-sdk-1.1.3
```

   To build a QIMP SDK layer, the following export is required:

```
export EXTRALAYERS="meta-qcom-qim-product-sdk"
```

3. Set up the build environment:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
```

```
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

4. Build the software image:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qim-product-sdk
```

On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
qcom-multimedia-image
ls -al system.img
```

Build QIRP SDK image

1. Download Qualcomm Yocto and supporting layers:

NOTE The `<manifest release tag>` for QIRP SDK build is the same as the base build. QIRP SDK layers must be cloned on top of the base build.

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
```

NOTE For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

2. Download the QIRP SDK layers into the base build `<WORKSPACE DIR>` directory:

```
git clone https://git.codelinaro.org/clo/le/meta-ros.git -b
ros.qclinux.1.0.r1-rel layers/meta-ros
git clone https://github.com/quic-yocto/meta-qcom-robotics.git layers/meta-
qcom-robotics
git clone https://github.com/quic-yocto/meta-qcom-robotics-distro.git
layers/meta-qcom-robotics-distro
git clone https://github.com/quic-yocto/meta-qcom-robotics-sdk.git layers/
meta-qcom-robotics-sdk
git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk layers/
meta-qcom-qim-product-sdk
```

3. Set up the build environment:

```
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=qcm6490 DISTRO=qcom-robotics-ros2-humble source setup-robotics-
```

```
environment
# source setup-robotics-environment: Sets the environment settings,
creates the build directory build-qcom-robotics-ros2-humble,
# and enters into build-qcom-robotics-ros2-humble directory.
```

4. Build the robotics image and QIRP SDK artifacts:

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

   On a successful build, you can see the QIRP SDK build artifacts at the following paths:

   ```
   QIRP SDK artifacts: <WORKSPACE DIR>/build-qcom-robotics-ros2-humble/
   tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk_<version>.tar.gz
   # system.img is present in the following path
   Robotics image: <WORKSPACE DIR>/build-qcom-robotics-ros2-humble/tmp-
   glibc/deploy/images/qcm6490/qcom-robotics-full-image
   ```

Build real-time Linux image

1. Download Qualcomm Yocto and supporting layers:

   NOTE The `<manifest release tag>` for real-time Linux image is the same as the base build. Real-time Linux must be cloned on top of the base build.

   ```
   # cd to directory where you have 300 GB of free storage space to create
   your workspaces
   mkdir <WORKSPACE_DIR>
   cd <WORKSPACE_DIR>
   repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
   kirkstone -m <manifest release tag>
   # Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
   repo sync
   ```

   NOTE For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

2. Clone real-time Linux layer into the workspace:

   ```
   git clone https://github.com/quic-yocto/meta-qcom-realtime -b <meta-qcom-
   realtime release tag> layers/meta-qcom-realtime
   # Example, <meta-qcom-realtime release tag> is qcom-6.6.28-QLI.1.1-
   Ver.1.1_realtime-linux-1.0
   ```

   To build a real-time layer, the following export is required:

   ```
   export EXTRALAYERS="meta-qcom-realtime"
   ```

3. Set up the build environment:

   ```
   MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
   # source setup-environment: Sets the environment settings, creates the
   build directory build-qcom-wayland,
   # and enters into build-qcom-wayland directory
   ```

4. Build the software image:

   ```
   bitbake qcom-multimedia-image
   ```

On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
qcom-multimedia-image
ls -al system.img
```

Flash

Flash software images to the device using Flash images for registered users.

5.5 Build with Dockerfile

Ubuntu host setup

- Install git:

  ```
  # Install git if you have not already installed
  sudo apt install git
  ```

- Clone the `qcom-download-utils` git repository, which provides Dockerfile for Qualcomm public Yocto layers and a few helper scripts:

  ```
  mkdir <workspace_path>
  cd <workspace_path>
  git clone https://git.codelinaro.org/clo/le/qcom-download-utils.git
  cd qcom-download-utils
  ```

- Add user to the Docker group:

  ```
  sudo usermod -aG docker $USER
  newgrp docker
  # To check if user is part of a Docker group, run the following command:
  sudo grep /etc/group -e "docker"
  ```

Check the machine configuration

- Check if your host is configured accurately:

  ```
  bash utils/check_config.sh
  # Resolve the errors and run this command until no errors show up
  ```

- Install Docker:

  ```
  bash docker/docker_setup.sh
  ```

  NOTE As part of the Docker setup, the `qcom-download-utils` directory structure is shown in the following figure:

```
├── docker
│   ├── docker_build.sh
│   ├── dockerfiles
│   │   ├── Dockerfile_20.04
│   │   └── Dockerfile_22.04
│   ├── docker_run.sh
│   └── docker_setup.sh
├── LICENSE
├── qcom-6.6.00-QLI.1.0-Ver.1.1
│   └── config.sh
├── qcom-6.6.13-QLI.1.0-Ver.1.2
│   └── config.sh
├── qcom-6.6.13-QLI.1.0-Ver.1.3
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.3
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.3_qim-product-sdk-1.1
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.3_realtime-linux-1.0
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.4
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.4_qim-product-sdk-1.1
│   └── config.sh
├── qcom-6.6.17-QLI.1.0-Ver.1.4_realtime-linux-1.0
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.0
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.0_qim-product-sdk-1.1.1
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.1
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3
│   └── config.sh
├── qcom-6.6.28-QLI.1.1-Ver.1.1_realtime-linux-1.0
│   └── config.sh
├── README.md
└── utils
├── check_config.sh
└── sync_build.sh
```

Build base image

Create a Yocto Docker image and build:

1. Run `docker_build.sh` to create the Docker image with Dockerfile (**Dockerfile_22.04**) and Dockertag (**qcom-6.6.28-qli.1.1-ver.1.1_22.04**). This Docker image is used to create the container environment to run the Yocto build.

   Dockertag: Release folder in lower case letters appended by the Dockerfile OS version to enable easy identification of the release build with Dockerfile (Docker does not allow upper case letters in the Dockertag).

   NOTE See Docker for troubleshooting Docker issues.

   ```
   bash docker/docker_build.sh -f ./docker/dockerfiles/Dockerfile_22.04 -t
   qcom-6.6.28-qli.1.1-ver.1.1_22.04
   ```

   If you face any issues while running `docker_build.sh`, see the following solution:

   ```
   # Error 1: cache related issue.
      # If you are facing issue with the docker build command, try using
   --no-cache option. This option will force rebuilding of layers already
   available
      $ bash docker/docker_build.sh -n --no-cache -f ./docker/dockerfiles/
   Dockerfile_22.04 -t qcom-6.6.28-qli.1.1-ver.1.1_22.04

   # Error2: response from daemon: Get "https://
   registry-1.docker.io/v2/": http: server gave HTTP response to HTTPS
   client
      # Add internal Docker registry mirror. (Internal-only setting for
   the Qualcomm network)
      # Using a tab instead of space and other invisible white-space
   characters may break the proper work of json configuration files
      # and later may lead to Docker service failing to start.

      # Solution:
        sudo vim /etc/docker/daemon.json
        # Add an entry similar to the following in /etc/docker/
   daemon.json:
        {
           "registry-mirrors": ["https://docker-registry.qualcomm.com"]
        }
        # Restart the Docker service to take the new settings
        sudo systemctl restart docker
   ```

2. Sync and build the Yocto image in a Docker container with the Docker tag and release parameters:

   ```
   bash docker/docker_run.sh -t qcom-6.6.28-qli.1.1-ver.1.1_22.04 -r
   qcom-6.6.28-QLI.1.1-Ver.1.1
   ```

   Build workspace is available in `<qcom-download-utils download path>/<release>/build-qcom-wayland`. For example, `qcom-download-utils/qcom-6.6.28-QLI.1.1-Ver.1.1/build-qcom-wayland`.

   NOTE **# ERROR: error.GitError: git config ('--replace-all', 'color.ui', 'auto'): error: could not write config file /home/$USER/.gitconfig: Device or resource busy**

This error is triggered when your gitconfig does not set the UI color configuration as Git 1.8.4 is enabled by default. To enable color display in your account, run the following command: `git config --global color.ui auto`.

NOTE If a build error is triggered and fixed, run the commands in Rebuild.

Build QIMP SDK image

1. Build base image with Docker.
2. Build QIMP SDK on top of the base image with Docker:
   a. Run the `docker run` command:

```
# Run the following commands inside the base image build location
cd <workspace_path>/qcom-download-utils/qcom-6.6.28-QLI.1.1-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-6.6.28-qli.1.1-ver.1.1_22.04 /bin/bash
```

   b. Clone QIMP SDK layer into the workspace:

```
git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk -b
<qim-product-sdk release tag> layers/meta-qcom-qim-product-sdk
# Example, <qim-product-sdk release tag> is qcom-6.6.28-QLI.1.1-
Ver.1.1_qim-product-sdk-1.1.3
```

   To build a QIMP SDK layer, the following export is required:

```
export EXTRALAYERS="meta-qcom-qim-product-sdk"
```

   c. Set up the build environment:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory
```

   d. Build the software image:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qim-product-sdk
```

Rebuild

- List the Docker images:

```
docker images
```

  This command generates the following output:

```
REPOSITORY
TAG                          IMAGE ID       CREATED        SIZE
qcom-6.6.28-qli.1.1-ver.1.1_22.04
latest                       8fcea388d8ca   2 days ago     1.47GB
```

- Attach the container:

```
# Run the following commands outside the Docker container
cd <workspace_path>/qcom-download-utils/qcom-6.6.28-QLI.1.1-Ver.1.1

# Run the following commands inside the base image build location
bash
docker run -it -v "${HOME}/.gitconfig":"/home/${USER}/.gitconfig" -v "$
{HOME}/.netrc":"/home/${USER}/.netrc" -v $(pwd):$(pwd) -w $(pwd)
qcom-6.6.28-qli.1.1-ver.1.1_22.04 /bin/bash

# Example
WORKSPACE=<workspace_path>/qcom-download-utils/qcom-6.6.28-QLI.1.1-Ver.1.1
```

- Set up the build environment:

```
# cd <release directory>
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory
```

- Build the software image:

```
bitbake qcom-multimedia-image
```

Flash

Flash software images to the device using Flash images for registered users.

6 GitHub workflow (firmware and extras)

NOTE These steps are applicable only for authorized users. To upgrade your access, go to http://www.qualcomm.com/support/working-with-qualcomm.

To build various Qualcomm Linux Yocto layers including `meta-qcom-hwe`, `meta-qcom-extras`, `meta-qcom-qim-product-sdk`, and `meta-qcom-robotics-sdk` using selective proprietary sources and binaries/libraries, follow these step-by-step reference instructions.

6.1 Host machine requirements

Configuration	Tools	Permissions
x86 machine	Git 1.8.3.1 or later versions	A `sudo` permission is required to execute a few commands.
Quad-core CPU, for example, Intel i7-2600 at 3.4 GHz (equivalent or better)	Tar 1.28 or later versions	
300 GB free disk space (swap partition > 32 GB)	Python 3.10.2 or later versions	
16 GB RAM	GCC 7.5 or later versions	
Ubuntu 22.04	GNU Make 4.0 or later versions	

6.2 Install QSC CLI

Set up `qsc-cli`:

1. Install curl (if not already installed):

   ```
   sudo apt install curl
   ```

2. Download the Debian package for `qsc-cli`:

   ```
   cd <workspace_path>
   curl -L https://softwarecenter.qualcomm.com/api/download/software/qsc/linux/latest.deb -o qsc_installer.deb
   ```

3. Install the `qsc-cli` Debian package:

   ```
   sudo dpkg -i qsc_installer.deb
   ```

4. Log in to `qsc-cli`:

```
qsc-cli login -u <username>
```

NOTE A one-time login into chipcode.qti.qualcomm.com is required to download Qualcomm proprietary git repositories. Use your Qualcomm login credentials to complete this step.

For more details, see `qsc-cli` related topics in How to.

6.3 Ubuntu host setup

The Ubuntu host machine needs a few setup operations to ensure that the required software tools are ready.

1. Install the following packages:

```
sudo apt update
sudo apt install repo gawk wget git diffstat unzip texinfo gcc build-
essential chrpath socat cpio python3 python3-pip python3-pexpect xz-utils
debianutils iputils-ping python3-git python3-jinja2 libegl1-mesa libsdl1.2-
dev pylint xterm python3-subunit mesa-common-dev zstd liblz4-tool locales
tar python-is-python3 file libxml-opml-simplegen-perl vim whiptail
sudo apt-get install lib32stdc++6 libncurses5 checkinstall libreadline-dev
libncursesw5-dev libssl-dev libsqlite3-dev tk-dev libgdbm-dev libc6-dev
libbz2-dev libffi-dev curl
```

2. Add your Qualcomm login ID with PAT to the `.netrc` file in your home directory:

```
# Log in to qsc-cli to generate PAT
qsc-cli login -u <username>
# Run the following command to generate PAT
qsc-cli pat --get
# This command gives output as shown in the following note
# The last line in this output is the token, which can be used to access
# Qualcomm Proprietary repositories. This token expires in two weeks.
```

NOTE user@hostname:/local/mnt/workspace$ qsc-cli pat --get

[Info]: Starting qsc-cli version 0.0.0.9

5LThNIkIb55mMVLB5C2KqUGU2jCF

```
vim ~/.netrc # add the following entries

machine chipmaster2.qti.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>

machine qpm-git.qualcomm.com
login <your Qualcomm login id>
password <your PAT token>
```

3. Set up the locales (if not set up already):

```
sudo locale-gen en_US.UTF-8
sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
export LC_ALL=en_US.UTF-8
export LANG=en_US.UTF-8
```

4. Update git configurations:

```
# Check if your identity is configured in .gitconfig
git config --get user.email
git config --get user.name

# Run the following commands if you do not have your account identity set
in .gitconfig
git config --global user.email <Your email ID>
git config --global user.name <"Your Name">

# Add the following UI color option for output of console (optional)
git config --global color.ui auto

# Add the following git configurations to fetch large size repositories
and to avoid unreliable connections
git config --global http.postBuffer 1048576000
git config --global http.maxRequestBuffer 1048576000
git config --global http.lowSpeedLimit 0
git config --global http.lowSpeedTime 999999

# Add the following git configurations to follow remote redirects from
http-alternates files or alternates
git config --global http.https://
chipmaster2.qti.qualcomm.com.followRedirects true
git config --global http.https://qpm-git.qualcomm.com.followRedirects true
```

5. Set up Python 3.10.2:

NOTE Skip the following instructions if you already have python 3.10.2 or later versions.

```
python --version
# Download it in a directory of your choice
wget https://www.python.org/ftp/python/3.10.2/Python-3.10.2.tgz
tar -xvf Python-3.10.2.tgz
cd Python-3.10.2
# Use sudo if you need to access /opt
./configure --prefix=/opt/python3 --enable-optimizations
make
make install
ln -s /opt/python3/bin/python3  /opt/python3/bin/python
export PATH=/opt/python3/bin:$PATH
export PYTHONPATH=/opt/python3/lib:$PYTHONPATH
```

6.4 Build with firmware sources

Sync firmware

The following table describes the Qualcomm Yocto layers and release tags:

Table 6-1 Qualcomm Yocto layers and manifest tags

Access level	Yocto layer	Release tag	Example
Registered developer with any email address	meta-qcom-hwe	manifest release tag	qcom-6.6.28-QLI.1.1-Ver.1.1.xml
	meta-qcom-qim-product-sdk	manifest release tag	qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3.xml
	meta-qcom-robotics-sdk	manifest release tag	qcom-6.6.28-QLI.1.1-Ver.1.1_robotics-product-sdk-1.1.xml
Registered developer from a verified organization	meta-qcom-extras	meta-qcom-extras release tag	r1.0_00041.0
See Table 6-2.	NA	firmware release tag	r1.0_00039.2

The following tables describe the available distributions for firmware that can be downloaded according to the need and entitlements:

Table 6-2 Mapping access levels

Access level	Distribution	Yocto layers
Registered developer from a verified organization	Base build: High-level OS and firmware source (GPS only) Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem	meta-qcom meta-qcom-hwe meta-qcom-extras
	Base build + QIMP SDK Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIMPSDK	meta-qcom meta-qcom-hwe meta-qcom-extras meta-qcom-qim-product-sdk
	Base build + QIMP SDK + QIRP SDK 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

Table 6-2 Mapping access levels (cont.)

Access level	Distribution	Yocto layers

Licensed developer with additional access	Base build: High-level OS and firmware (GPS only) source
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

	Base build + QIMP SDK (GPS only)
	`Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

	Base build: High-level OS and firmware (GPS and modem) source
	`Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`

	Base build + QIMP SDK (GPS and modem)
	`Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|QIMPSDK`
		`meta-qcom`
		`meta-qcom-hwe`
		`meta-qcom-extras`
		`meta-qcom-qim-product-sdk`

For Yocto layer descriptions, see Table 2-1.

Table 6-3 Mapping of firmware distributions and git repositories

Firmware distribution	Git command	Directory into which firmware gets synced on git clone
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_nomodem.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_nomodem`
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIMPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk`
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NM_QIRPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk`
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem.git`	`qualcomm-linux-spf-1-0_ap_standard_oem`

Table 6-3 Mapping of firmware distributions and git repositories (cont.)

Firmware distribution	Git command	Directory into which firmware gets synced on git clone
Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|QIMPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_qimpsdk.git`	`qualcomm-linux-spf-1-0_ap_standard_oem_qimpsdk`
Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_amss_standard_oem.git`	`qualcomm-linux-spf-1-0_amss_standard_oem`
Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|QIMPSDK	`git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-spf-1-0_amss_standard_oem_qimpsdk.git`	`qualcomm-linux-spf-1-0_amss_standard_oem_qimpsdk`

NOTE Commands in the following sections are based on binary and source for firmware images without modem and GPS (see the command in Table 6-3). Hence, `qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk` is used. If you use any other distribution, then update the directory accordingly.

The **Git command** column (see Table 6-3) provides the git repository, which contains the firmware sources. These repositories are hosted on Qualcomm servers. Clone the appropriate repositories based on your access profile and use case. The following git clone command downloads the selected firmware components in source, except the modem:

```
mkdir -p <FIRMWARE_ROOT>
cd <FIRMWARE_ROOT>
git clone -b <firmware release tag> --depth 1 https://qpm-git.qualcomm.com/
home2/git/qualcomm/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk.git
# Example, <firmware release tag> is r1.0_00039.2
```

NOTE The `git clone` command clones the content into the `<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk` directory.

For the latest `<firmware release tag>`, see the *Build-critical release tags* section in the Release Notes.

Build firmware

- **Prerequisites**
  - Ensure that the working shell is `bash`. Check the output:

    ```
    echo $0
    ```

    The expected output of the command should be `bash`. If not, enter the bash shell:

    ```
    bash
    ```

- Install libffi6 using the following commands. This is required for the QAIC compiler, which generates header and source files from IDL files:

```
curl -LO http://archive.ubuntu.com/ubuntu/pool/main/libf/libffi/
libffi6_3.2.1-8_amd64.deb
sudo dpkg -i libffi6_3.2.1-8_amd64.deb
```

- Install LLVM for AOP, TZ, and BOOT compilation:

```
cd <FIRMWARE_ROOT>
mkdir llvm

# Log in to qpm-cli and activate the license
qpm-cli --login
qpm-cli --license-activate sdllvm_arm

# LLVM requirement for BOOT compilation is 10.0.3
qpm-cli --install sdllvm_arm --version 10.0.3 --path <FIRMWARE_ROOT>/
llvm/10.0.3
chmod -R 777 <FIRMWARE_ROOT>/llvm/10.0.3

# LLVM requirement for AOP is 14.0.4
qpm-cli --install sdllvm_arm --version 14.0.4 --path <FIRMWARE_ROOT>/
llvm/14.0.4
chmod -R 777 <FIRMWARE_ROOT>/llvm/14.0.4

# LLVM requirement for TZ compilation is 16.0.7
qpm-cli --install sdllvm_arm --version 16.0.7 --path <FIRMWARE_ROOT>/
llvm/16.0.7
chmod -R 777 <FIRMWARE_ROOT>/llvm/16.0.7
```

- Export the `SECTOOLS` variable and compile the firmware builds (`<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk` is the top-level directory):

```
export SECTOOLS=<FIRMWARE_ROOT>/qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qimpsdk/<product>/common/sectoolsv2/ext/
Linux/sectools
export SECTOOLS_DIR=<FIRMWARE_ROOT>/qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qimpsdk/<product>/common/sectoolsv2/ext/Linux
# An example <product> is QCM6490.LE.1.0, see the latest Release Notes.
```

- Install and set up Qualcomm® Hexagon™:

```
qpm-cli --extract hexagon8.4 --version 8.4.07
export HEXAGON_ROOT=$HOME/Qualcomm/HEXAGON_Tools
echo $HEXAGON_ROOT
```

NOTE Set the environment variable HEXAGON_ROOT to the path where the Hexagon SDK is installed. To change the install path when using `qpm-cli`, see How can I change the Hexagon tool install path?.

- **Build cDSP**

  Tools required

  - Compiler version – Hexagon 8.4.07
  - Python version – Python 3.10.2
  - Install libffi6

  Build steps

  a. Navigate to the following directory:

  ```
  cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
  CDSP.HT.2.5.c3/cdsp_proc/build/ms
  ```

  b. Clean the build:

  ```
  python ./build_variant.py kodiak.cdsp.prod --clean
  ```

  c. Build the image:

  ```
  python ./build_variant.py kodiak.cdsp.prod
  ```

- **Build aDSP**

  Tools required

  - Compiler version – Hexagon 8.4.07
  - Python version – Python 3.10.2
  - Install libffi6

  Nanopb integration (only one-time setup)

  ```
  cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
  ADSP.HT.5.5.c8/adsp_proc/qsh_api
  curl https://jpa.kapsi.fi/nanopb/download/nanopb-0.3.9.5-linux-x86.tar.gz -
  o nanopb-0.3.9.5-linux-x86.tar.gz
  cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
  ADSP.HT.5.5.c8/adsp_proc/
  python qsh_api/build/config_nanopb_dependency.py -f nanopb-0.3.9.5-linux-
  x86
  ```

  Build steps

  a. Navigate to the following directory:

  ```
  cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
  ADSP.HT.5.5.c8/adsp_proc/build/ms
  ```

  b. Clean the build:

  ```
  python ./build_variant.py kodiak.adsp.prod --clean
  ```

  c. Build the image:

  ```
  python ./build_variant.py kodiak.adsp.prod
  ```

- **Build Boot**

  Tools required

  - Compiler version – LLVM version must be updated to 10.0.3

    NOTE To avoid build errors, ensure that there is a `/` at the end of the command.

    ```
    export LLVM=<FIRMWARE_ROOT>/llvm/10.0.3/
    ```

  - Python version – Python 3.10
  - Install libffi6

  Build steps

  a. For compiling Boot, you need the device tree compiler in the `/pkg/qct/software/boottools` directory. Install the package:

  ```
  sudo apt-get install device-tree-compiler
  ```

  b. Copy the `/usr/bin/dtc` file to the `/pkg/qct/software/boottools` directory:

  ```
  sudo mkdir -p /pkg/qct/software/boottools
  sudo cp -r /usr/bin/dtc /pkg/qct/software/boottools
  ```

  c. Navigate to the following directory:

  ```
  cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
  BOOT.MXF.1.0.c1/
  ```

  d. Install the dependencies:

  ```
  python -m pip install -r boot_images/boot_tools/dtschema_tools/oss/
  requirements.txt
  ```

  e. Clean the build:

  ```
  python -u boot_images/boot_tools/buildex.py -t kodiak,QcomToolsPkg -v
  LAA -r RELEASE --build_flags=cleanall
  ```

  f. Build the image:

  ```
  python -u boot_images/boot_tools/buildex.py -t kodiak,QcomToolsPkg -v
  LAA -r RELEASE
  ```

  NOTE For debug variant builds, replace `RELEASE` with `DEBUG`.

- **TZ firmware**

  Tools required

  - Compiler version – LLVM 16.0.7
  - Python version – Python 3.10

  Build steps

  a. Install LLVM:

  ```
  cd <FIRMWARE_ROOT>/TZ.XF.5.0/trustzone_images/build/ms/
  vi build_config_deploy_kodiak.xml
  # Edit all the occurrences of /pkg/qct/software/llvm/release/arm/
  16.0.7/ to <FIRMWARE_ROOT>/llvm/16.0.7/
  ```

b. Build the image:

```
cd <FIRMWARE_ROOT>/TZ.XF.5.0/trustzone_images/build/ms/
python build_all.py -b TZ.XF.5.0 CHIPSET=kodiak --
cfg=build_config_deploy_kodiak.xml
```

- **AOP firmware**

  Tools required

  - Compiler version – LLVM 14.0.4
  - Python version – Python 3.10

  Build steps

  a. Navigate to the following directory:

  ```
  cd <FIRMWARE_ROOT>/AOP.HO.3.6/aop_proc/build/
  ```

  b. Build the image:

  ```
  ./build_kodiak.sh -l <FIRMWARE_ROOT>/llvm/14.0.4/
  ```

- **CPUCP firmware**

  The CPUCP firmware is released as a binary and build compilation is not required.

- **CPUSYS.VM firmware**

  The CPUSYS.VM firmware is released as a binary and build compilation is not required.

- **BTFM firmware**

  The BTFM firmware is released as a binary and build compilation is not required.

- **WLAN firmware**

  The WLAN firmware is released as a binary and build compilation is not required.

- **Generate firmware prebuilds (boot-critical and split-firmware binaries)**

  Create an integrated firmware binary from the individual components that you compiled:

  ```
  # cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
  <product>/common/build
  # An example <product> is QCM6490.LE.1.0, see the latest Release Notes
  cd <FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/
  QCM6490.LE.1.0/common/build
  python build.py --imf
  ```

  NOTE Firmware prebuild is successful if the following zip files are generated in the `<FIRMWARE_ROOT>/qualcomm-linux-spf-1-0_ap_standard_oem_nm-qimpsdk/QCM6490.LE.1.0/common/build/ufs/bin` directory:

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

Build base image with extras

1. Download Qualcomm Yocto and supporting layers with extras:

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
git clone https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-
spf-1-0_hlos_oem_metadata.git -b <meta-qcom-extras release tag> --depth 1
# Example, <meta-qcom-extras release tag> is r1.0_00041.0
mkdir -p layers/meta-qcom-extras
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/QCM6490.LE.1.0/common/
config/meta-qcom-extras/* layers/meta-qcom-extras/
```

NOTE For the `<manifest release tag>` and `<meta-qcom-extras release tag>` information, see the *Build-critical release tags* section in the Release Notes.

2. Set up the Yocto build:

```
# Export additional meta layers to EXTRALAYERS. Location is assumed under
<WORKSPACE DIR>/layers.
export EXTRALAYERS="meta-qcom-extras"

# CUST_ID is used to clone the proprietary source repositories downloaded
by meta-qcom-extras.
# It enables source compilation for the corresponding binaries present in
meta-qcom-hwe.
# This ID is constant for the firmware repository qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qirpsdk.git.
# CUST_ID must be initialized to <PARTY_ID> for "Licensed customers with
additional access".
# For example, for distros like "Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|"
and "Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|",
# <PARTY_ID> is provided while signing up for distros mapping to "Licensed
customers with additional access".
# To find <PARTY_ID>, sign in to your account at qualcomm.com.
# Click the profile icon, select Account Settings, and then scroll to
Company Information section.
# Use the number specified for Export ID as <PARTY_ID>.
export CUST_ID="213195"

# The firmware recipe is compiled when the Yocto build is initiated.
Firmware recipe expects the
# path of firmware. You have generated firmware prebuilts (boot-critical
and split-firmware binaries)
# using the steps described in the previous section. The directory path
```

```
must contain QCM6490_bootbinaries.zip,
# QCM6490_dspso.zip, and QCM6490_fw.zip.
# Set the environment variable to pick up the prebuilts:
export FWZIP_PATH="<FIRMWARE_ROOT>/qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qimpsdk/<product>/common/build/ufs/bin"
# An example <product> is QCM6490.LE.1.0. For more information on
<product>, see the latest Release Notes.

MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

3. Compile the Yocto build:

```
bitbake qcom-multimedia-image
```

NOTE Clean the Yocto build:

```
bitbake -fc cleansstate qcom-multimedia-image
bitbake -fc cleanall qcom-multimedia-image
```

On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
qcom-multimedia-image
ls -al system.img
```

4. Flash the generated build using Flash images for registered users.

Build QIMP SDK image with extras

1. Download QIMP SDK layers, Qualcomm Yocto, and supporting layers with extras:

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
git clone https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-
spf-1-0_hlos_oem_metadata.git -b <meta-qcom-extras release tag> --depth 1
# Example, <meta-qcom-extras release tag> is r1.0_00041.0
mkdir -p layers/meta-qcom-extras
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/QCM6490.LE.1.0/common/
config/meta-qcom-extras/* layers/meta-qcom-extras/
git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk -b
```

```
qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3 layers/meta-qcom-qim-
product-sdk
```

NOTE For the <manifest release tag> and <meta-qcom-extras release tag> information, see the *Build-critical release tags* section in the Release Notes.

2. Set up the Yocto build:

```
# Export additional meta layers to EXTRALAYERS. Location is assumed under
<WORKSPACE DIR>/layers.
export EXTRALAYERS="meta-qcom-extras meta-qcom-qim-product-sdk"

# CUST_ID is used to clone the proprietary source repositories downloaded
by meta-qcom-extras.
# It enables source compilation for the corresponding binaries present in
meta-qcom-hwe.
# This ID is constant for the firmware repository qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qirpsdk.git.
# CUST_ID must be initialized to <PARTY_ID> for "Licensed customers with
additional access".
# For example, for distros like "Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|"
and "Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|",
# <PARTY_ID> is provided while signing up for distros mapping to "Licensed
customers with additional access".
# To find <PARTY_ID>, sign in to your account at qualcomm.com.
# Click the profile icon, select Account Settings, and then scroll to
Company Information section.
# Use the number specified for Export ID as <PARTY_ID>.
export CUST_ID="213195"

# The firmware recipe is compiled when the Yocto build is initiated.
Firmware recipe expects the
# path of firmware. You have generated firmware prebuilts (boot-critical
and split-firmware binaries)
# using the steps described in the previous section. The directory path
must contain QCM6490_bootbinaries.zip,
# QCM6490_dspso.zip, and QCM6490_fw.zip.
# Set the environment variable to pick up the prebuilts:
export FWZIP_PATH="<FIRMWARE_ROOT>/qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qimpsdk/<product>/common/build/ufs/bin"
# An example <product> is QCM6490.LE.1.0. For more information on
<product>, see the latest Release Notes.

MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the
build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory.
```

3. Compile the QIMP SDK build:

```
bitbake qcom-multimedia-image
bitbake qim-product-sdk
```

NOTE Clean the QIMP SDK build:

```
bitbake -fc cleansstate qcom-multimedia-image
bitbake -fc cleanall qcom-multimedia-image

bitbake -fc cleansstate qim-product-sdk
bitbake -fc cleanall qim-product-sdk
```

On successful build, you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
qcom-multimedia-image
ls -al system.img
```

4. Flash the generated build using Flash images for registered users.

Build QIRP SDK image with extras

NOTE Ensure that you have cloned the respective firmware for QIRP SDK. For example, `qualcomm-linux-spf-1-0_ap_standard_oem_nm-qirpsdk`.

1. Download QIRP SDK layers, Qualcomm Yocto, and supporting layers with extras:

```
# cd to directory where you have 300 GB of free storage space to create
your workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1.xml
repo sync
git clone https://qpm-git.qualcomm.com/home2/git/qualcomm/qualcomm-linux-
spf-1-0_hlos_oem_metadata.git -b <meta-qcom-extras release tag> --depth 1
# Example, <meta-qcom-extras release tag> is r1.0_00041.0
mkdir -p layers/meta-qcom-extras
mkdir -p layers/meta-qcom-robotics-extras
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/QCM6490.LE.1.0/common/
config/meta-qcom-extras/* layers/meta-qcom-extras/
cp -rf qualcomm-linux-spf-1-0_hlos_oem_metadata/QCM6490.LE.1.0/common/
config/meta-qcom-robotics-extras/* layers/meta-qcom-robotics-extras/

git clone https://git.codelinaro.org/clo/le/meta-ros.git -b
ros.qclinux.1.0.r1-rel layers/meta-ros
git clone https://github.com/quic-yocto/meta-qcom-robotics.git layers/meta-
qcom-robotics
git clone https://github.com/quic-yocto/meta-qcom-robotics-distro.git
```

```
layers/meta-qcom-robotics-distro
git clone https://github.com/quic-yocto/meta-qcom-robotics-sdk.git layers/
meta-qcom-robotics-sdk
git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk -b <qim-
product-sdk release tag> layers/meta-qcom-qim-product-sdk
# Example, <qim-product-sdk release tag> is qcom-6.6.28-QLI.1.1-
Ver.1.1_qim-product-sdk-1.1.3
```

NOTE For the <manifest release tag>, <meta-qcom-extras release tag>, and <qim-product-sdk release tag> information, see the *Build-critical release tags* section in the Release Notes.

2. Set up the Yocto build:

```
# Export additional meta layers to EXTRALAYERS. Location is assumed under
<WORKSPACE DIR>/layers.
export EXTRALAYERS="meta-qcom-extras meta-qcom-robotics-extras"

# CUST_ID is used to clone the proprietary source repositories downloaded
by meta-qcom-extras.
# It enables source compilation for the corresponding binaries present in
meta-qcom-hwe.
# This ID is constant for the firmware repository qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qirpsdk.git.
# CUST_ID must be initialized to <PARTY_ID> for "Licensed customers with
additional access".
# For example, for distros like "Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|"
and "Qualcomm_Linux.SPF.1.0|AMSS|Standard|OEM|",
# <PARTY_ID> is provided while signing up for distros mapping to "Licensed
customers with additional access".
# To find <PARTY_ID>, sign in to your account at qualcomm.com.
# Click the profile icon, select Account Settings, and then scroll to
Company Information section.
# Use the number specified for Export ID as <PARTY_ID>.
export CUST_ID="213195"

# The firmware recipe is compiled when the Yocto build is initiated.
Firmware recipe expects the
# path of firmware. You have generated firmware prebuilts (boot-critical
and split-firmware binaries)
# using the steps described in the previous section. The directory path
must contain QCM6490_bootbinaries.zip,
# QCM6490_dspso.zip, and QCM6490_fw.zip.
# Set the environment variable to pick up the prebuilts:
export FWZIP_PATH="<FIRMWARE_ROOT>/qualcomm-linux-
spf-1-0_ap_standard_oem_nm-qirpsdk/<product>/common/build/ufs/bin"
# An example <product> is QCM6490.LE.1.0. For more information on
<product>, see the latest Release Notes.
```

3. Compile the QIRP SDK build:

```
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=qcm6490 DISTRO=qcom-robotics-ros2-humble source setup-robotics-environment
# source setup-robotics-environment: Sets the environment settings, creates the build directory build-qcom-robotics-ros2-humble,
# and enters into build-qcom-robotics-ros2-humble directory.
../qirp-build qcom-robotics-full-image
```

   On a successful build, you can see the QIRP SDK build artifacts at the following paths:

```
QIRP SDK artifacts: <workspace_path>/build-qcom-robotics-ros2-humble/tmp-glibc/deploy/qirpsdk_artifacts/qirp-sdk_<version>.tar.gz
# system.img is present in the following path
Robotics image: <workspace_path>/build-qcom-robotics-ros2-humble/tmp-glibc/deploy/images/qcm6490/qcom-robotics-full-image
```

4. Flash the generated build using Flash images for registered users.

7 Flash images for unregistered users

Unregistered users must flash the software using the following steps:

1. Update `udev` rules (one-time prerequisite).
2. Move the device to Emergency Download (EDL) mode.
3. Flash software using QDL tool.

Update `udev` rules

Ensure that the `udev` USB rules for the Qualcomm manufacturing vendor ID **05c6** are configured on the Linux host:

1. Navigate to the `udev` USB rules directory:

   ```
   cd /etc/udev/rules.d
   ```

2. List the contents of the directory:

   ```
   ls
   ```

   - If the `51-qcom-usb.rules` file is not present, use `sudo vi 51-qcom-usb.rules` to create it and add the following content to the file:

     ```
     SUBSYSTEMS=="usb", ATTRS{idVendor}=="05c6", ATTRS{idProduct}=="9008",
     MODE="0666", GROUP="plugdev"
     ```

   - If the file exists, then check for the previous content:

     ```
     cat51-qcom-usb.rules
     ```

3. Restart `udev`:

   ```
   sudo systemctl restart udev
   ```

If the USB cable is already connected to the host, unplug the cable and then reconnect it for the updated rules to take effect.

Move to EDL mode

The device must be in EDL mode before you flash the software. The Qualcomm RB3 Gen 2 platform enters EDL mode if there is no image on the device after power up or if it is corrupted. To force the device into EDL mode, use any one of the following methods:

Using UART

NOTE The RB3 Gen 2 device must have a Qualcomm Linux build image.

1. Connect device to UART shell.
2. Move the device to EDL mode:

   ```
   reboot edl
   ```

3. Verify whether the device has entered the QDL mode:

   ```
   lsusb
   ```

   Sample output

   ```
   Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem
   (QDL mode)
   ```

   NOTE This procedure is verified on an Ubuntu host machine and not on a VM.

Using ADB

NOTE The RB3 Gen 2 device must have a Qualcomm Linux build image.

1. Install QUD on the host device.
2. Install ADB on the host device.
3. Connect ADB to the RB3 Gen 2 device.
4. Move the device to EDL mode:

   ```
   adb shell reboot edl
   ```

5. Verify whether the device has entered the QDL mode:

   ```
   lsusb
   ```

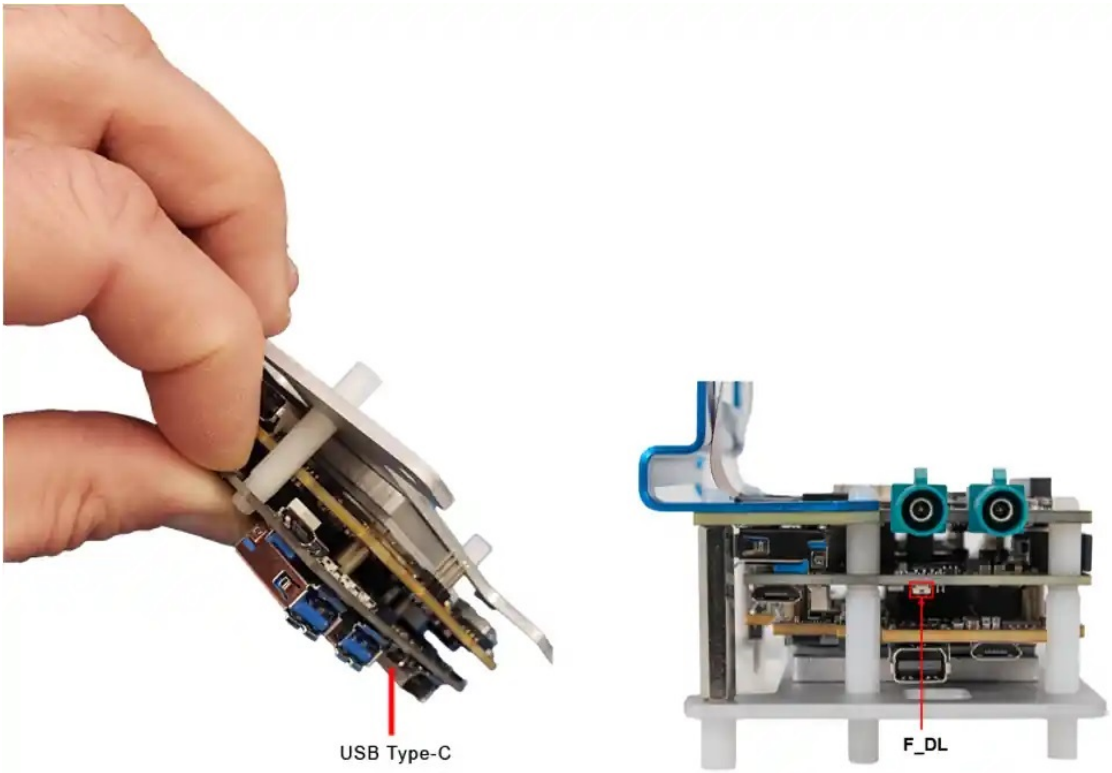
   Sample output

   ```
   Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem
   (QDL mode)
   ```

   NOTE This procedure is verified on an Ubuntu host machine and not on a VM.

Manual

1. Press and hold the **F_DL** button:



2. Connect the device to a +12 V wall power supply.
3. Connect the device to the host system using a Type-C cable through the USB Type-C connector.
4. Release the **F_DL** button. The device should now be in Qualcomm download (QDL) mode. For this task, QDL is used interchangeably with EDL.
5. Verify whether the device has entered the QDL mode:

   ```
   lsusb
   ```

   Sample output

   ```
   Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem
   (QDL mode)
   ```

Flash software using QDL

1. Ensure ModemManager is not running.

   Some Linux distributions come with ModemManager, a tool for configuring mobile broadband. When the device is connected in USB mode, it is identified as a Qualcomm modem, and ModemManager tries to configure the device. As this interferes with QDL flashing, you must disable ModemManager before connecting your device. If you are using a Linux distribution with `systemd`, then ModemManager can be stopped using the following command:

   ```
   sudo systemctl stop ModemManager
   ```

   If you need ModemManager, you can start it again after the flashing is complete.

2. Navigate to the following location in the workspace to find the QDL tool and flash the images:

```
# Usage:
# cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/<MACHINE>/
<IMAGE>
# build_path: For DISTRO=qcom-wayland, it is build-qcom-wayland.
#             For DISTRO=qcom-robotics-ros2-humble, it is build-qcom-
robotics-ros2-humble
# qdl <prog.mbn> [<program> <patch> ...]
# Example: build_path is build-qcom-wayland
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/
qcom-multimedia-image
./qdl prog_firehose_ddr.elf rawprogram*.xml patch*.xml
```

Flashing is successful if you see *partition 1 is now bootable* on the terminal window as shown in the following message:

```
LOG: INFO: Calling handler for setbootablestoragedrive
LOG: INFO: Using scheme of value = 1
partition 1 is now bootable
LOG: INFO: Calling handler for power
LOG: INFO: Will issue reset/power off 100 useconds, if this hangs
check if watchdog is enabled
LOG: INFO: bsp_target_reset() 1
```

After a successful flashing operation, run the `lsusb` command to see the device information on the terminal window as shown in line 4 of the following message:

```
ThinkPad-T490s:<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/
images/qcm6490/qcom-multimedia-image$ lsusb
Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
Bus 002 Device 006: ID 05c6:901d Qualcomm, Inc. QCM6490_fd2913cf
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
```

NOTE If flashing fails, perform the following steps and retry the flashing procedure:

1. Power off the device.
2. Disconnect from the host.
3. Reboot the host.

Do not move the QDL tool from this location to another alternate path or host PC. If you must use the standalone QDL, see How to build a standalone QDL?.

To connect to the device, see How to SSH.

NOTE The device reboots on successful completion of the flashing procedure. To verify the updated software version, see Check software version.

8 Flash images for registered users

Registered users must flash the software using the following steps:

1. Install PCAT and QUD tools on host machine (one-time prerequisite).
2. Update `udev` rules (one-time prerequisite).
3. Move the device to EDL mode.
4. Flash the software using QDL, PCAT, or QSC Launcher.

Install PCAT and QUD

To detect connected devices and flash software builds, ensure that the Qualcomm PCAT and QUD tools are installed on the host machine. Run the following commands to use `qpm-cli` to install PCAT and QUD:

```
qpm-cli --login
qpm-cli --install pcat --activate-default-license
qpm-cli --install qud --activate-default-license
```

NOTE For Ubuntu 22.04, you may encounter an issue while installing QUD, where you might be asked to enroll the public key on your Linux host for a successful QUD installation. For additional details, follow the steps provided in the README file available in the `/opt/QUIC/sign/signReadme.txt` directory.

Update `udev` rules

See Update `udev` rules.

Move to EDL mode

See Move to EDL mode.

Flash software using QDL

See Flash software using QDL.

Flash software using PCAT

1. Check if `QTI_HS-USB_QDLoader` driver is available in the installed directory:

   ```
   ls -la /dev/Q*
   ```

   Sample output

```
crw-rw-rw- 1 root 242 0 Dec 10 10:51 /dev/QTI_HS-
USB_QDLoader_9008_3-8:1.0
```

2. Verify whether the device has entered the QDL mode:

   ```
   lsusb
   ```

   Sample output

   ```
   Bus 002 Device 014: ID 05c6:9008 Qualcomm, Inc. Gobi Wireless Modem
   (QDL mode)
   ```

3. Check if the device is recognized by PCAT:

   ```
   pcat -devices
   ```

   Sample output

   ```
   Searching devices in Device Manager, please wait for a moment…
   ID | DEVICE TYPE | DEVICE STATE | SERIAL NUMBER | ADB SERIAL NUMBER |
   DESCRIPTION
   NA | NA          | EDL          | BE116704      | be116704          |
   Qualcomm USB Composite Device:QUSB_BULK_CID:042F_SN:BE116704
   ```

4. Download the build:

   ```
   PCAT -PLUGIN SD -DEVICE <device_serial_number> -BUILD
   "<build_images_path>" -MEMORYTYPE UFS -FLAVOR asic

   # Example
   PCAT -PLUGIN SD -DEVICE be116704 -BUILD "<workspace_path>/build-qcom-
   wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image" -MEMORYTYPE
   UFS -FLAVOR asic
   ```

   Flashing is successful if you see the following message:

   ```
   xxxx INFO] [ FIRMWARE DOWNLOAD LOG ] Process
   Finished
   xxxx INFO] Status   - TRUE
   xxxx INFO] Response - Downloading software images completed on the
   device Qualcomm USB Composite Device:QUSB_BULK_CID:042F_SN:BE116704
   ```

   After a successful flashing operation, run `lsusb` command to see the device information on the terminal window as shown in line 4 of the following message:

   ```
   ThinkPad-T490s:<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/
   images/qcm6490/qcom-multimedia-image$ lsusb
   Bus 004 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
   Bus 003 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
   Bus 002 Device 006: ID 05c6:901d Qualcomm, Inc. QCM6490_fd2913cf
   Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
   ```

NOTE

- To connect to the device, see How to SSH.
- The device reboots on successful completion of the flashing procedure. To verify the updated software version, see Check software version.

Flash software using QSC Launcher

See Flash.

NOTE The device reboots on successful completion of the flashing procedure. To verify the updated software version, see Check software version.

9 Troubleshooting

Docker

- ***docker: Cannot connect to the Docker daemon at unix:///var/run/docker.sock. Is the docker daemon running?***

  Run the following command to start Docker:

  ```
  sudo systemctl start docker
  ```

- ***Error response from daemon: Get "https://registry-1.docker.io/v2/": http: server gave HTTP response to HTTPS client***

  Add an internal Docker registry mirror (internal setting for Qualcomm network).

  NOTE Do not include # comments in the JSON configuration file.

  Using a tab instead of space and other invisible whitespace characters may break the proper work of JSON configuration files and may lead to `docker.service` failing to start.

  ```
  sudo vim /etc/docker/daemon.json
  # Add an entry similar to the following in /etc/docker/daemon.json:
  {
          "registry-mirrors": ["https://docker-registry.qualcomm.com"]
  }
  ```

  Restart the Docker service to take the new settings.

  ```
  sudo systemctl restart docker
  ```

- ***Docker failure due to Virtualization not enabled***

  This error can be resolved by enabling virtualization from the BIOS. Follow the specific instructions from the system provider to enable the virtualization. For example, the following steps can enable virtualization provided by a system provider:

  a. When the system is booting up, step into the BIOS. The **BIOS** window is displayed.

  b. Switch to the **Advanced** tab.

  c. In the **CPU Configuration** section, set **Virtualization Technology** to enabled.

  d. Save and exit.

  e. Restart the system.

- ***Permission denied while trying to connect to the Docker daemon socket at unix:///var/run/docker.sock***

  This happens when `qsc-cli` is already installed on the machine and the user is not part of the Docker group:

  ```
  sudo usermod -aG docker $USER
  newgrp docker
  ```

  To confirm that you are part of the Docker group, run the following command:

  ```
  sudo grep /etc/group -e "docker"
  # This command shows a list of users who are part of the Docker group;
  must include your user ID
  ```

  Log out and log in again for the access to take effect.

  ```
  # You can run the following command to check if you are part of the Docker
  group
  id -a
  # This command returns an output string which should include 'docker'
  ```

Sync

- ***repo init or sync failure with except ManifestInvalidRevisionError, e:***

  You might encounter this issue after installing the Repo package:

  - If you have redirection in your `/etc/gitconfig` or `~/.gitconfig` to an internal mirror.
  - If your internal mirror has prefix to branches while mirroring.

  For example, if `/etc/gitconfig` redirects and the internal mirror has a stable branch from upstream git mirrored as `aosp/stable`, then the following error appears while performing `repo init`:

  ```
  [url "ssh://<internal mirror>:<port>/tools/repo"]
    insteadOf = https://android.googlesource.com/tools/repo
    insteadOf = https://gerrit.googlesource.com/git-repo
  ```

  ```
  File "/local/mnt/workspace/<userid>/test_repo/.repo/repo/main.py",
  line 126
      except ManifestInvalidRevisionError, e:
             ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^
  SyntaxError: multiple exception types must be parenthesized
  ```

  The steps to resolve this error are as follows:

  ```
  # Remove the older .repo folder. This will be in the directory where you
  ran 'repo init' command earlier
  rm -rf .repo

  # Export and run the repo commands to fix the repo issues. The REPO_REV
  must point to the mirrored
  # branch from upstream 'stable' branch of https://gerrit.googlesource.com/
  git-repo
  export REPO_REV='aosp/stable'
  ```

- ***Install repo "Server certificate verification failed"***

  If you see a certificate error such as ‘Server certificate verification failed. CAfile: none CRLfile: none’, configure git to disable SSL certificate verification with git configuration. Discuss with your IT administration for further guidance. You can use either of the following commands to disable SSL:

  ```
  export GIT_SSL_NO_VERIFY=1
  git config --global http.sslverify false
  ```

  If your region is blocking access to `android.googlesource`, try the following configuration to fetch Repo from Codelinaro Mirror:

  ```
  git config --global url.https://git.codelinaro.org/clo/la/tools/
  repo.insteadOf https://android.googlesource.com/tools/repo
  ```

- ***error.GitError: git config ('--replace-all', 'color.ui', 'auto'): error: could not write config file / home/$USER/.gitconfig: Device or resource busy***

  This error occurs when your gitconfig does not set the UI color configuration. This configuration is set by default in Git 1.8.4 and later versions. Run the following command to enable the color display in your account:

  ```
  git config --global color.ui auto
  ```

- ***[Error]: Failed preparing build for compilation. Error: Error setting docker credentials. Error: "Error saving credentials: error closing temp file: close /usr2/<userid>/.docker/ config.json3322274803: disk quota exceeded\n"***

  QSC CLI uses the home directory only for a few kilo bytes (kB). Clear a few MBs from your home directory.

- ***[Error]: The "path" argument must be of type string. Received undefined***

  Error excerpt

  ```
  qsc-cli download --workspace-path '/local/mnt/workspace/<userid>/K2L/
  QSC_CLI_BUILD/build' --product 'QCM6490.LE.1.0' --build
  'QCM6490.LE.1.0-00134-STD.PROD-1' --distribution
  'Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PUBLIC'
  [Info]: Starting qsc-cli version 0.0.0.7
  (node:2924765) ExperimentalWarning: The Fetch API is an experimental
  feature. This feature could change at any time
  (Use `qsc-cli --trace-warnings ...` to show where the warning was
  created)
  [Info]: Checking if Workspace already exists
  [Info]: Saved updated Workspace info
  [Info]: Workspace Setup Completed
  [Error]: The "path" argument must be of type string. Received undefined
  ```

  Solution

  This error occurs because QSC-CLI is incompatible with Qlauncher. Qlauncher is going to be deprecated and replaced with a new application from the QSC. If you have Qlauncher in the workspace, you can run the following commands:

  ```
  # Find your workspace within the Qlauncher UI
  # Take a backup of the following metadata file if you want to preserve the
  older workspace created with Qlauncher.
  # These will work only with Qlauncher app. You can reinstall the app at a
  later time again to access. If you do not
  ```

```
# need the workspaces, you can delete this file using:
mv /var/lib/qcom/data/qualcomm_launcher/workspaces2.json /var/lib/qcom/
data/qualcomm_launcher/workspaces2.json.bak
# Uninstall Qlauncher with the following command:
qpm-cli --uninstall qualcomm_launcher
```

- ***docker: Error response from daemon: error while creating mount source path '/usr2/<userid>/.netrc': mkdir /usr2/<userid>/.netrc: permission denied***

  Error excerpt

```
Updating files: 100% (64/64), done.
2024-02-29T07:58:00: Sync Command Completed
2024-02-29T07:58:01: Finished setup.
[Info]: Setting Docker Credential
2024-02-29T07:58:03: Authorization successful
2024-02-29T07:58:03: Sync Command Starting
2024-02-29T07:58:03: Running Sync Command for SyncOpenSourceCode -
DownloadOpenSource
docker: Error response from daemon: error while creating mount source
path '/usr2/ramevelp/.netrc': mkdir /usr2/ramevelp/.netrc: permission
denied.
2024-02-29T07:58:04: Sync Command Failed
[Error]: Failed SP Download with error: 2024-02-29T07:58:04: Sp
Download failed. ExitCode: 126 Signal: 0  with errorcode 4
[Error]: 2024-02-29T07:58:04: Sp Download failed. ExitCode: 126
Signal: 0
```

  Solution

  This could happen due to the way IT has set up your home directory. Work with your IT administrator for any further changes to your home directory.

- ***fatal: couldn't find remote ref refs/heads/qcom-linux$_{STX}$kirkstone***

  If you see any junk characters while copying commands from the PDF, remove or replace the junk characters with appropriate symbols and rerun the command. Alternately, you can open the guide in HTML mode and use the copy command option.

  Example

```
# Replace the following command
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-
linux$_{STX}$kirkstone -m qcom-6.6.28-QLI.1.1-Ver.1.1.xml
# with
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m qcom-6.6.28-QLI.1.1-Ver.1.1.xml
```

Build

- ***ERROR: linux-kernel-qcom-6.6-r0 do_menuconfig: No valid terminal found, unable to open devshell***

  This error might be triggered while running a `bitbake linux-kernel-qcom -c menuconfig` command.

  Error excerpt

```
ERROR: linux-kernel-qcom-6.6-r0 do_menuconfig: No valid terminal
found, unable to open devshell.
```

```
Tried the following commands:
        tmux split-window -c "{cwd}" "do_terminal"
        tmux new-window -c "{cwd}" -n "linux-kernel-qcom
Configuration" "do_terminal"
        xfce4-terminal -T "linux-kernel-qcom Configuration" -e
"do_terminal"
        terminology -T="linux-kernel-qcom Configuration" -e do_terminal
```

Solution

```
sudo apt install screen
sudo apt install tmux
```

- ***NOTE: No reply from server in 30s***

  If you are seeing this error during the build on the rerun of `qsc-cli compile` or `bitbake` commands, you can try to delete `bitbake.lock`, `bitbake.sock`, and `hashserve.lock` from your partially built workspace and retry the build. For example, if you are building with `qsc-cli`, then these files are found under `<absoute_workspace_path>/DEV/LE.QCLINUX.1.0.r1/build-qcom-wayland`.

- ***do_fetch: BitBake Fetcher Error: FetchError('Unable to fetch URL from any source')***

  These are intermittent fetch failures. Check if there is a network/host issue at your end, else it might be the server creating this issue. You could increase `postBuffer` and `maxRequestBuffer` settings in your `.gitconfig` file if the errors occur while fetching the git repositories. If you are using `qsc-cli`, then these configurations are already taken care by the `qsc-cli` tool:

  ```
  git config --global http.postBuffer 1048576000
  git config --global http.maxRequestBuffer 1048576000
  ```

  If these configurations do not work, you can retry the compile to get past these intermittent errors for the first time.

  A few large git projects may show this error. For such projects, a feasible option is to manually clone as follows:

  ```
  cd <workspace_path>/downloads/git2/
  git clone --bare --mirror https://<url>/<project-name>.git
  <workspace_path>/downloads/git2/<local-name>.git
  touch <workspace_path>/downloads/git2/<local-name>.git.done
  ```

  For example, when `do_fetch` fails for `qualcomm_linux-spf-1-0-le-qclinux-1-0-r1_api-linux_history_prebuilts.git`, run the following command:

  ```
  git clone --bare --mirror https://qpm-git.qualcomm.com/home2/git/revision-
  history/qualcomm_linux-spf-1-0-le-qclinux-1-0-r1_api-
  linux_history_prebuilts.git <workspace_path>/downloads/git2/qpm-
  git.qualcomm.com.home2.git.revision-history.qualcomm_linux-spf-1-0-le-
  qclinux-1-0-r1_api-linux_history_prebuilts.git
  touch <workspace_path>/downloads/git2/qpm-
  git.qualcomm.com.home2.git.revision-history.qualcomm_linux-spf-1-0-le-
  qclinux-1-0-r1_api-linux_history_prebuilts.git.done
  ```

  After creating the `.done` file, proceed with the `bitbake <image-recipe>` command. After the first build completes, it is recommended to set up your own download directory.

- ***make[4]: /bin/sh: Argument list too long***

  This happens when the path of the workspace exceeds 90 characters. Reduce the workspace path length to avoid this failure.

- ***kernel-source/arch/arm64/boot/dts/qcom/qcm6490-idp.dts:8:10: fatal error: dt-bindings/iio/qcom,spmi-adc7-pmk8350.h: No such file or directory***

  The file in question `qcom,spmi-adc7-pmk8350.h` is part of the kernel source `<kernel-src>/include/dt-bindings/iio/qcom,spmi-adc7-pmk8350.h`.

  Check the workspace for this file and ensure that the environment is properly initialized to pick this file. While compiling dtbs, the kernel build system runs a GCC preprocessor to replace the macros in dts files by its definition. The mentioned path is one such place where several `includes` reside.

  Check if you have `core.symlinks` set to `false` in your git configuration. If yes, set it to true:

  ```
  git config --global core.symlinks true
  ```

- ***qpm-git.qualcomm.com.home2.git.revision-history.qualcomm_linux-spf-1-0-le-qclinux-1-0-r1_api-linux_history_prebuilts.git --progress failed with exit code 128, no output***

  128 is a masking error and this error needs further triage as it could be a network issue at your end or a genuine issue accessing Qualcomm or upstream mirrors. As a workaround for this error, see do_fetch: BitBake Fetcher Error: FetchError('Unable to fetch URL from any source'). You can triage it further by following the subsequent instructions to dump verbose logs during fetch.

  By default, verbose logging is not enabled for Yocto git fetch. To enable the same for all git projects, edit `local.conf` file and change `BB_GIT_VERBOSE_FETCH` value to **1**. Verbose logging can also be enabled for each recipe. For example, to enable verbose logging and debug a `do_fetch()` failure in a diag recipe, perform the following steps:

  a. Edit `layers/meta-qcom-hwe/recipes-bsp/diag/daig_15.0.bb` and add the line `BB_GIT_VERBOSE_FETCH = "1"`.

  b. Clean the previous downloaded artifacts using `bitbake -fc cleanall diag`.

  c. Fetch the source again using `bitbake -fc fetch diag`.

  d. Fetch the log with verbose logging available under the diag recipe's working directory `build-qcom-wayland/tmp-glibc/work/qcm6490-qcom-linux/diag/15.0-r0/temp`.

  e. Share `log.do_fetch` from this path with the Qualcomm CE point of contact.

  NOTE Enabling git verbose logging for all recipes can significantly increase the build time. It is recommended to enable it only in required recipes on a need basis.

- ***Failed SP Download with error: <> Sp Download failed. ExitCode: 128 Signal: 0 with errorcode 4***

  Error excerpt

  ```
  warning: redirecting to https://git-na-ssl.chipcode.qti.qualcomm.com/
  57f0ec058e47f7a82b2de7b95111c74a/qualcomm/qualcomm-linux-
  spf-1-0_ap_standard_oem_nomodem.git/
  remote: Counting objects: 129803, done.
  remote: Compressing objects: 100% (114948/114948), done.
  fatal: write error: No space left on device5 GiB | 1.63 MiB/s
  fatal: fetch-pack: invalid index-pack output
  2024-03-02T14:32:18: Sync Command Failed
  [Error]: Failed SP Download with error: 2024-03-02T14:32:18: Sp
  Download failed. ExitCode: 128 Signal: 0 with errorcode 4
  ```

```
[Error]: 2024-03-02T14:32:18: Sp Download failed. ExitCode: 128
Signal: 0
```

Solution

Error log indicates that there is no space on the device**"fatal: write error: No space left on device"**.

Clean up the space and retrigger.

- ***File "/usr/lib/python3.10/locale.py", line 620, in setlocale return _setlocale(category, locale)locale.Error: unsupported locale setting***

  To resolve this error, run the following commands and recompile:

  ```
  sudo locale-gen en_US.UTF-8
  sudo update-locale LC_ALL=en_US.UTF-8 LANG=en_US.UTF-8
  export LC_ALL=en_US.UTF-8
  export LANG=en_US.UTF-8
  ```

- ***layer directories do not exist build-qcom-wayland/conf/../../layers/meta-qcom-qim-product-sdk***

  This error occurs due to one of the following reasons:

  - Git clone of `meta-qcom-qim-product-sdk` did not complete successfully.
  - `meta-qcom-qim-product-sdk` layer is not exported to EXTRALAYERS.

  Error excerpt

  ```
  xxxx@xxxx:~/github_un/build-qcom-wayland$ bitbake qcom-multimedia-image
  ERROR: The following layer directories do not exist:
  ERROR: <workspace_path>/build-qcom-wayland/conf/../../layers/meta-qcom-
  qim-product-sdk
  ERROR: Please check BBLAYERS in <workspace_path>/build-qcom-wayland/
  conf/bblayers.conf
  ```

  Solution

  - Remove the `build-qcom-wayland` directory.
  - Rerun the commands in Build QIMP SDK image.

- ***failed: database disk image is malformed. abort()ing pseudo client by server request***

  The Pseudo tool encounters path mismatch and corrupt database issues when processing file system operations. When Pseudo simulates file system operations in a Yocto project, problems might occur in the process of handling file paths and permissions.

  This is a known issue in the Yocto community.

  Solution

  Run the following commands:

  ```
  rm -rf <workspace_path>/build-qcom-robotics-ros2-humble/tmp-glibc
  bitbake -c cleanall pseudo-native & bitbake pseudo-native
  ../qirp-build qcom-robotics-full-image
  ```

Flash

No known errors.

10 How to

10.1 Sync

Repo installation alternate methods

The latest Repo works with python3 and if your default Python is pointed to python2, then install `python-is-python3` to make python3 as the default Python.

```
mkdir -p ~/bin
cd ~/bin
# If you already have a previous directory of repo_tool, you can delete it
rm -rf ~/bin/repo_tool
git clone https://android.googlesource.com/tools/repo.git -b v2.41 repo_tool
cd repo_tool
git checkout -b v2.41
export PATH=~/bin/repo_tool:$PATH
```

If the previous steps do not work, you can install Repo using the following commands:

```
# Install curl (if it is not installed)
sudo apt install curl bc

# Latest Repo version works with python3

mkdir -p ~/bin
curl https://raw.githubusercontent.com/GerritCodeReview/git-repo/v2.41/repo -
o ~/bin/repo && chmod +x ~/bin/repo
export PATH=~/bin:$PATH
```

How does QSC-CLI work?

1. **Setup**

   QSC-CLI installs Docker and sets up the necessary git configuration.

2. **Sync**

   QSC-CLI downloads the firmware sources as needed based on the input parameters and completes the Qualcomm Yocto layers build.

3. **Build**

   QSC-CLI builds the Qualcomm firmware as needed and also completes the build for the Qualcomm Yocto layers.

4. Internally, QSC-CLI implements the standalone commands covered in the GitHub workflow (firmware and extras) and leverages the prebuilt Docker images for the respective Qualcomm style software images. For example, `LE.QCLINUX.1.0.r1`.

How to see more information on commands and options supported by QSC-CLI?

To see all the commands provided by QSC-CLI, run the following commands:

```
qsc-cli -h
qsc-cli download -h
```

To see more details about a particular command, you can append `-h` to the command. For example:

```
qsc-cli compile -h
```

How to manage workspaces using QSC-CLI?

List the workspaces using the following command:

```
qsc-cli workspace info --list
```

To delete a workspace, run the following command:

```
qsc-cli workspace delete --workspace-path <workspace_path>

# Example
qsc-cli workspace delete --workspace-path '/local/mnt/worskspace/
Qworkspace_QIMPSDK'
```

How do I find my Yocto workspace with QSC-CLI?

You can install the `tree` command and run it on your workspace. The Yocto workspace is under the `LE.QCLINUX.1.0.r1` directory as shown in the following views. These directories stay the same for future releases.

- **QSC-CLI workspace structure after `Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK` distribution build**

  The following is a sample view, in which:

  - `LE.QCLINUX.1.0.r1` contains the Yocto workspace.
  - Remaining directories can be ignored as all the necessary sources and binaries are encoded in the Yocto layer recipes synced under `LE.QCLINUX.1.0.r1/layers`.

```
├── about.html
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY22
│   │   ├── snap_release.xml
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── build-qcom-wayland
│   │   ├── bitbake-cookerdaemon.log
│   │   ├── bitbake.lock
│   │   ├── bitbake.sock
│   │   ├── buildhistory
│   │   ├── cache
│   │   ├── conf
│   │   ├── qim-prod-sdk
│   │   ├── qim-sdk
│   │   ├── tflite-sdk
│   │   └── tmp-glibc
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-hwe
│   │   ├── meta-qcom-qim-product-sdk
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       └── snap_release.xml
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa.gsifw
│   │   └── sectoolsv2
│   └── contents.xml
```

- **QSC-CLI workspace structure after `Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem` distribution build with firmware and extras**

  The following is a sample view, in which:

  - `LE.QCLINUX.1.0.r1` contains the Yocto workspace.
  - Few additional directories are for the Qualcomm firmware. While building with extras:
    - These additional firmware are built.
    - The output binaries from these are picked up from firmware recipes in the Qualcomm Yocto layers.
    - For detailed sync and build instructions, see GitHub workflow (firmware and extras).

```
├── about.html
├── ADSP.HT.5.5.c8
│   ├── adsp_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_kodiak.adsp.prodQ.json
├── AOP.HO.3.0.2
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── AOP.HO.3.6
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
├── BTFW.HSP.2.1.2
│   └── btfw_proc
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
├── CDSP.HT.2.5.c3
│   ├── BuildProducts.txt
│   ├── cdsp_proc
│   └── VariantImgInfo_kodiak.cdsp.prodQ.json
├── CPUCP.FW.1.0
│   └── cpucp_proc
├── DSP.AT.1.0
│   └── dsp_proc
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   ├── build-qcom-wayland
│   ├── downloads
│   ├── layers
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   ├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
│   └── sstate-cache
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   └── contents.xml
├── QCS9100.LE.1.0
│   ├── common
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
├── TZ.APPS.1.0
│   ├── qtee_tas
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   ├── BuildProducts.txt
│   ├── HY22
│   └── trustzone_images
├── WLAN.HSP.1.1
│   └── wlan_proc
└── WLAN.MSL.2.0.c2
    └── wlan_proc
```

How to refresh the workspace with a new download using QSC CLI?

This option is supported only for the `LE.QCLINUX.1.0.r1` image, which syncs the Yocto layers and prepares the Yocto workspace to be built. This includes the following steps:

NOTE Get to a Docker shell as mentioned in How to generate eSDK?.

1. Download a new release:

```
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-kirkstone -m <release tag>
repo sync
```

NOTE For the `<manifest release tag>` information, see the *Build-critical release tags* section in the Release Notes. An example `<manifest release tag>` is `qcom-6.6.28-QLI.1.1-Ver.1.1.xml`.

2. Set up the build environment:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
```

3. Build the software image:

```
bitbake qcom-multimedia-image
```

10.2 Build

How do I know if my build is completed?

If your build instruction is `bitbake qcom-multimedia-image`, then you can check if `system.img` is present in the `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image` directory:

```
cd <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image
ls -al system.img
```

How to generate eSDK?

Get a Docker shell and shell prompt

1. List the Docker images:

```
docker images
```

The output for this command is as follows:

```
REPOSITORY
TAG                          IMAGE ID       CREATED        SIZE
032693710300.dkr.ecr.us-west-2.amazonaws.com/stormchaser/ql-tool
20.04.20231220102843864.9    864b345bd707   2 months ago   715MB
032693710300.dkr.ecr.us-west-2.amazonaws.com/stormchaser/le.um-k2c
20.04.20231215014450998.7    4678dda58a91   2 months ago   929MB
```

2. Attach the container:

```
WORKSPACE=<WORKSPACE_PATH> && SRC_DIR=<SoftwareImage> && docker run --rm  -it -v
```

```
~/.qualcomm_launcher_workspace_config:/var/tmp/.docker_qualcomm_launcher_se
tup/ -v $WORKSPACE:$WORKSPACE -e LOCAL_USER_NAME=`id -u -n` -e
LOCAL_USER_ID=`id -u` -e USER=`id -u` -e WORKSPACE=$WORKSPACE -w
$WORKSPACE/$SRC_DIR <REPOSITORY:TAG> bash

# Example
WORKSPACE=/local/mnt/workspace/Qworkspace/DEV && SRC_DIR=LE.QCLINUX.1.0.r1
&& docker run --rm  -it -v
~/.qualcomm_launcher_workspace_config:/var/tmp/.docker_qualcomm_launcher_se
tup/ -v $WORKSPACE:$WORKSPACE -e LOCAL_USER_NAME=`id -u -n` -e
LOCAL_USER_ID=`id -u` -e USER=`id -u` -e WORKSPACE=$WORKSPACE -w
$WORKSPACE/$SRC_DIR 032693710300.dkr.ecr.us-west-2.amazonaws.com/
stormchaser/le.um-k2c:20.04.20231215014450998.7 bash
```

Check if you are in a workspace that has `.repo` in it. Set up the environment and generate eSDK:

NOTE When the eSDK generation build command is complete, the images are generated in the following directory: `<workspace_path>/build-qcom-wayland/tmp-glibc/deploy/sdk`.

1. After building `meta-qcom-hwe` with QSC CLI:

```
MACHINE=<machine> DISTRO=<distro> source setup-environment
bitbake -c do_populate_sdk_ext <image>

# Example
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

2. After building with `meta-qcom-extras` and firmware sources with QSC CLI:

   NOTE This step is not applicable for unregistered users.

```
# Example
export EXTRALAYERS="meta-qcom-extras"
export CUST_ID="213195"
export FWZIP_PATH="/local/mnt/workspace/extras/DEV/QCM6490.LE.1.0/common/
build/ufs/bin"
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

3. After building Standalone instructions within the same shell (shell where build is successful):

```
bitbake -c do_populate_sdk_ext <image>

# Example
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

4. After building with standalone instructions and with new shell (assuming build workspace exists):

```
cd <workspace_path>
MACHINE=<machine> DISTRO=<distro> source setup-environment
bitbake -c do_populate_sdk_ext <image>
```

```
# Example
cd /local/mnt/workspace/LE.QCLINUX.1.0.r1
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

5. After building with standalone instructions using Dockerfile.
   a. Move the control to workspace directory:

```
cd /local/mnt/workspace/qcom-download-utils/<release>

# Example
cd /local/mnt/workspace/qcom-download-utils/qcom-6.6.28-QLI.1.1-Ver.1.1
```

   b. Set up the environment and issue an eSDK build:

```
MACHINE=<machine> DISTRO=<distro> source setup-environment
bitbake -c do_populate_sdk_ext <image>

# Example
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
bitbake -c do_populate_sdk_ext qcom-multimedia-image
```

eSDK generation troubleshooting – basehash mismatch

Error excerpt

```
ERROR: When reparsing /local/mnt/workspace/extras/DEV/LE.QCLINUX.1.0.r1/
build-qcom-wayland/conf/../../layers/meta-qcom-distro/recipes-products/
images/qcom-multimedia-image.bb:do_populate_sdk_ext, the basehash value
changed from
7bce27b0510cb666f1bba1d03f055cfef48f9db2eabc17d490e14bbe4c632eba to
48ccd9d7370e0bf2435aa8b5067162932e07a3832adfa6ca037aa0ddb765c8de. The
metadata is not deterministic and this needs to be fixed.
ERROR: The following commands may help:
ERROR: $ bitbake qcom-multimedia-image -cdo_populate_sdk_ext -Snone
ERROR: Then:
ERROR: $ bitbake qcom-multimedia-image -cdo_populate_sdk_ext -Sprintdiff
```

Solution

Rebuild the image and generate eSDK again.

How to rebuild using Docker environment?

Following are the commands to connect to Docker for your environment setup and then use the BitBake commands to rebuild:

```
cd <workspace_path>/DEV/<softwareimage>
# Example, cd /local/mnt/workspace/Qworkspace/DEV/LE.QCLINUX.1.0.r1 for
making changes to Yocto layers
# Make code changes
```

NOTE Get to a Docker shell as mentioned in How to generate eSDK?.

- Rebuild using your source changes:

```
# Rebuild commands
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
bitbake qcom-multimedia-image
```

- Build image `qcom-multimedia-test-image`:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
bitbake qcom-multimedia-test-image
```

How to build a standalone QDL?

Prerequisites:

- The modules `make` and `gcc` must be available.
- Install the following dependent packages:

```
sudo apt-get install git libxml2-dev libusb-1.0-0-dev pkg-config
```

1. Download and compile the Linux flashing tool (QDL):

```
git clone --depth 1 --branch master https://github.com/linux-msm/qdl
cd qdl
git checkout cbd46184d33af597664e08aff2b9181ae2f87aa6
make
```

2. Flash using the generated QDL:

```
./qdl --storage ufs --include <workspace_path>/build-qcom-wayland/tmp-
glibc/deploy/images/qcm6490/qcom-multimedia-image <workspace_path>/build-
qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image/
prog_firehose_ddr.elf <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/
images/qcm6490/qcom-multimedia-image/rawprogram*.xml <workspace_path>/
build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image/
patch*.xml
```

How can I change the Hexagon tool install path?

`HEXAGON_ROOT` environment variable must point to the path where the Hexagon tools are installed. By default, `qpm-cli` installs `HEXAGON_ROOT` in `$HOME`. You can also choose an alternate directory to install `HEXAGON_ROOT`.

You can use the `--path` option in `qpm-cli` command to install Hexagon tools in a directory of your choice and export the `HEXAGON_ROOT` variable to the same directory.

Provide an absolute path for `<TOOLS_DIR>` in `qpm-cli` and export commands as shown in the following example:

```
# Example

mkdir -p <TOOLS_DIR>
qpm-cli --extract hexagon8.4 --version 8.4.07 --path <TOOLS_DIR>/8.4.07
export HEXAGON_ROOT=<TOOLS_DIR>
```

What are the image recipes supported as part of the GitHub workflow?

Image recipe	Description
qcom-minimal-image	A minimal rootfs image that boots to shell
qcom-console-image	Boot to shell with package group to bring in all the basic packages
qcom-multimedia-image	Image recipe includes recipes for multimedia software components, such as, audio, Bluetooth®, camera, computer vision, display, and video.
qcom-multimedia-test-image	Image recipe that includes tests

How to download the Platform eSDK?

1. Check the Host machine requirements.
2. Set up the Ubuntu host setup.
3. Download the Platform eSDK:
   a. Create a directory:

      ```
      mkdir <workspace_path>
      ```

   b. Open the directory:

      ```
      cd <workspace_path>
      ```

   c. Download the zipped file:
      - For Ubuntu x86 architecture-based host machines:

        ```
        wget https://artifacts.codelinaro.org/artifactory/qli-ci/flashable-binaries/qimpsdk/qcm6490/x86/qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3.zip
        ```

      - For Arm architecture-based host machines:

        ```
        wget https://artifacts.codelinaro.org/artifactory/qli-ci/flashable-binaries/qimpsdk/qcm6490/arm/qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3.zip
        ```

   d. Unzip the QIMP SDK to a directory of your choice:

      ```
      unzip qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3.zip
      ```

      After unzipping, ensure that the eSDK installer is located at <unzip_location>/target/qcm6490/sdk/:

      ```
      [/local/mnt/workspace/target/qcm6490/sdk]$ ls
      qcom-wayland-x86_64-qcom-multimedia-image-armv8-2a-qcm6490-toolchain-ext-1.0.host.manifest
      qcom-wayland-x86_64-qcom-multimedia-image-armv8-2a-qcm6490-toolchain-ext-1.0.sh
      qcom-wayland-x86_64-qcom-multimedia-image-armv8-2a-qcm6490-toolchain-ext-1.0.target.manifest
      qcom-wayland-x86_64-qcom-multimedia-image-armv8-2a-qcm6490-toolchain-ext-1.0.testdata.json
      x86_64-buildtools-nativesdk-standalone-1.0.host.manifest
      x86_64-buildtools-nativesdk-standalone-1.0.sh
      x86_64-buildtools-nativesdk-standalone-1.0.target.manifest
      x86_64-buildtools-nativesdk-standalone-1.0.testdata.json
      [/local/mnt/workspace/target/qcm6490/sdk]$
      ```

   e. If you do not have the necessary write permissions for the directory where you are trying to install the eSDK, the installer alerts you and then terminates. In such a scenario, set up the

permissions in the directory appropriately by using the following command and rerun the installer:

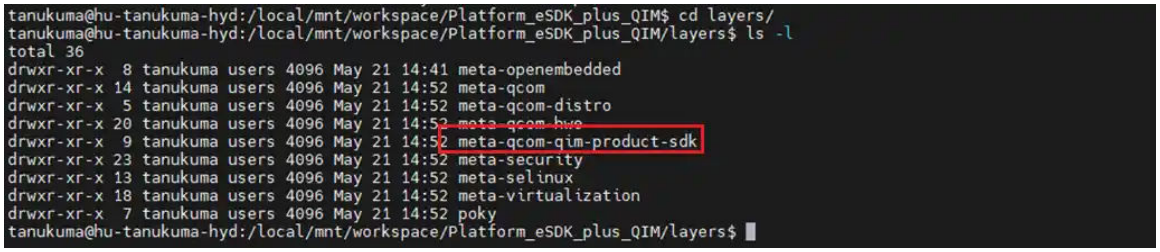
```
umask a+rx
```

4. Run the installer script to install the eSDK. For example:

   ```
   sh ./qcom-wayland-x86_64-qcom-multimedia-image-armv8-2a-qcm6490-toolchain-ext-1.0.sh
   ```

5. Follow the instructions on the console to install the eSDK in a convenient location of your host PC.
6. Ensure that the eSDK installation is successful.

   After installation, the QIMP SDK layers are included in the installation under `<workspace_path>/layers`:



   NOTE Advanced developers can still build their own eSDK by following the steps mentioned in Advanced procedure.

7. Run the following command to set the `ESDK_ROOT` variable:

   ```
   export ESDK_ROOT=<pathofinstallationdirectory>
   ```

   For example:

   ```
   export ESDK_ROOT=/local/mnt/workspace/Platform_eSDK_plus_QIM
   ```

The QIMP SDK installation is now complete. To develop an application for the platform, see Develop your first application.

10.3 Developer workflow

How to sync and build with real-time Linux?

`PREEMPT_RT` patches for the Qualcomm Linux kernel are provided in the `meta-qcom-realtime` layer. This layer is available in the Qualcomm github and it is built on top of the base build image. For more details on this layer, see meta-qcom-realtime from the Qualcomm Linux Yocto Guide.

To sync and build real-time Linux, see Build real-time Linux image.

NOTE Real-time kernel is not supported on QCS5430.

How to migrate from previous release to next release?

This depends on the development, branching, and integration workflows at your end. However, the following steps must be performed:

1. Integrate changes for the sources branched at your end:
   a. Skip this step if you have not forked any underlying source code used by Qualcomm Yocto layers and are applying only `.patch` files on top.
   b. Qualcomm is providing git history to all the source repositories. A reference list of repositories is provided in the Release Notes. For Qualcomm repositories that are branched and modified at your end, perform the following steps:
      i. Compare the Qualcomm Yocto layers with your Yocto layers, and identify the repositories you must migrate.
      ii. Check the Qualcomm Yocto layer recipes and identify the SRC_URI updates.
      iii. Clone these repositories using `git clone` and check out the appropriate branch and SHA as pointed by the SRC_URI.
      iv. Use `git merge` or appropriate mechanism to bring the delta to your own fork according to your need.
      v. Update your recipes as required.
2. Integrate the Yocto layer recipes – `git merge` the Yocto layer to pick up any changes.
3. Build and validate your resulting workspace.
4. Proceed with the next steps in your internal workflows.

How to build a Qualcomm Linux kernel?

See building the kernel.

How to customize Qualcomm Yocto layers?

See user customizations.

How to download layers for the QIMP SDK build using the manifest release tag

Download layers for the QIMP SDK:

NOTE For the latest `<manifest release tag>`, see the *Build-critical release tags* section in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to create your
workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-
```

```
sdk-1.1.3.xml
repo sync
```

NOTE For the steps to set up environment and build software images, see Build QIMP SDK image.

How to download layers for the QIRP SDK build by using manifest release tag

Download layers for the QIRP SDK:

NOTE For the latest <manifest release tag>, see the *Build-critical release tags* section in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to create your
workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1_robotics-
product-sdk-1.1.xml
repo sync
```

NOTE For the steps to set up environment and build software images, see Build QIRP SDK image.

How to download layers for the real-time Linux build by using manifest release tag

Download layers for the real-time Linux:

NOTE For the latest <manifest release tag>, see the *Build-critical release tags* section in the Release Notes.

```
# cd to directory where you have 300 GB of free storage space to create your
workspaces
mkdir <WORKSPACE_DIR>
cd <WORKSPACE_DIR>
repo init -u https://github.com/quic-yocto/qcom-manifest -b qcom-linux-
kirkstone -m <manifest release tag>
# Example, <manifest release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1_realtime-
linux-1.0.xml
repo sync
```

NOTE For the steps to set up environment and build software images, see Build real-time Linux image.

How to build a meta-qcom-qim-product-sdk layer as an add-on layer

If you have completed a base image build using GitHub standalone commands and already have an existing <WORKSPACE DIR>, you can follow these steps to build meta-qcom-qim-product-sdk:

1. Download the latest <meta-qcom-qim-product-sdk release tag>:

```
cd <WORKSPACE DIR>
rm -rf layers/meta-qcom-qim-product-sdk
git clone https://github.com/quic-yocto/meta-qcom-qim-product-sdk -b <meta-qcom-qim-product-sdk release tag> layers/meta-qcom-qim-product-sdk
```

   NOTE For more information on <meta-qcom-qim-product-sdk release tag>, see https://github.com/quic-yocto/meta-qcom-qim-product-sdk/tags. An example <meta-qcom-qim-product-sdk release tag> is qcom-6.6.28-QLI.1.1-Ver.1.1_qim-product-sdk-1.1.3.xml.

2. Set up the build environment:

```
MACHINE=qcm6490 DISTRO=qcom-wayland source setup-environment
# source setup-environment: Sets the environment settings, creates the build directory build-qcom-wayland,
# and enters into build-qcom-wayland directory
```

3. Build the software image:

```
bitbake qcom-multimedia-image
# Build SDK image
bitbake qim-product-sdk
```

10.4 Flash

How to flash Qualcomm configuration data table (CDT) with QDL?

For the Qualcomm hardware, the source of platform information on a device is the CDT, which is a blob that contains information about the current platform and its subvariants. The approach to update CDT is by flashing a CDT binary.

Following are the steps to flash CDT:

1. Download the CDT binary.
   - For the Core Kit, download the following file:

     ```
     wget https://artifacts.codelinaro.org/artifactory/codelinaro-le/qcom-device-config/CORE_KIT_CDT.bin
     ```

   - For the Vision Kit, download the following file:

     ```
     wget https://artifacts.codelinaro.org/artifactory/codelinaro-le/qcom-device-config/VISION_KIT_CDT.bin
     ```

2. Use the CDT binary.

   Copy the downloaded CDT binary to a compiled build location. The following is a sample command:

   ```
   cp <the downloaded CDT bin file> <workspace_path>/build-qcom-wayland/tmp-glibc/deploy/images/qcm6490/qcom-multimedia-image/
   ```

   NOTE `<workspace_path>` is the directory where you made your Yocto build.

3. Update the CDT section in `rawprogram3.xml` with the intended CDT filename. The following is a sample command:

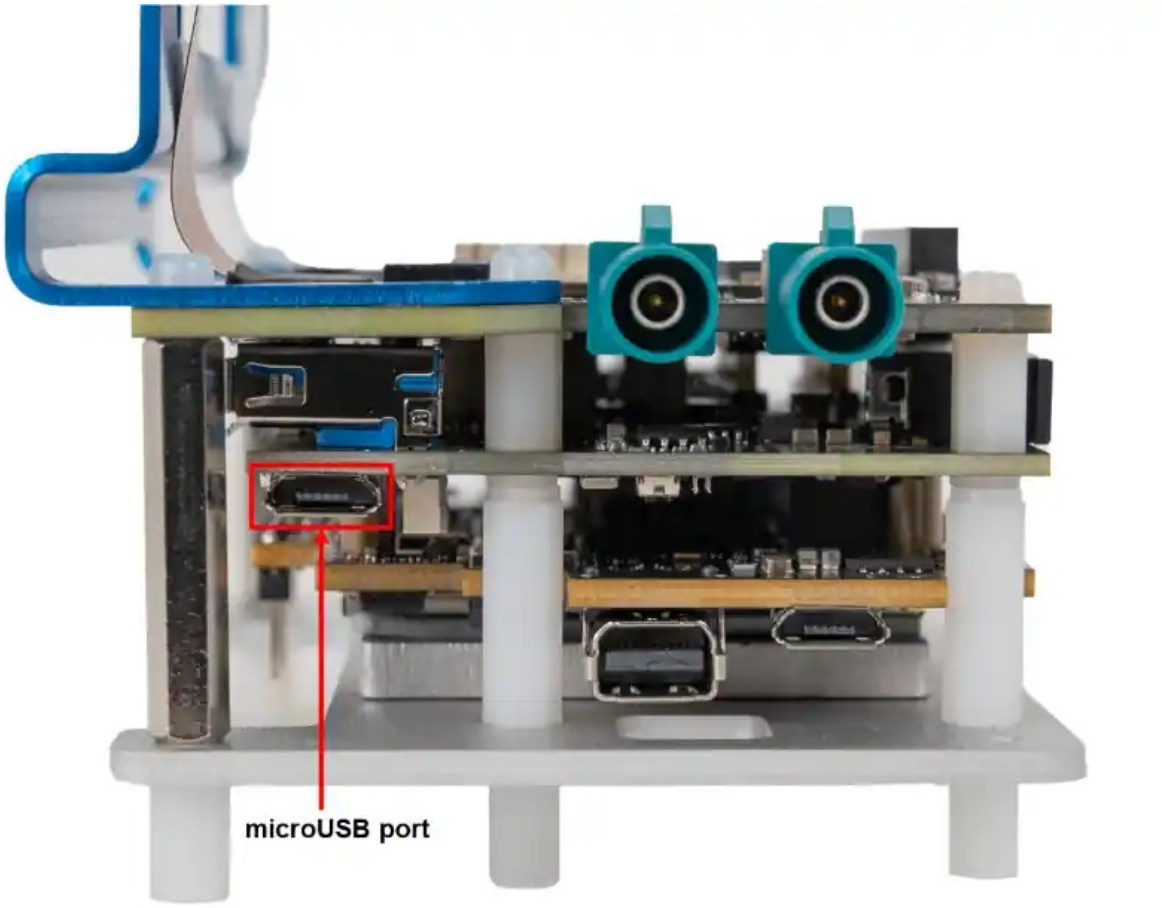
   ```
   <program SECTOR_SIZE_IN_BYTES="4096" file_sector_offset="0"
   filename="CORE_KIT_CDT.bin" label="cdt" num_partition_sectors="32"
   partofsingleimage="false" physical_partition_number="3"
   readbackverify="false" size_in_KB="128.0" sparse="false"
   start_byte_hex="0x20000" start_sector="32"/>
   ```

Continue with the flashing instructions.

10.5 Setup

How to connect to a UART shell?

To set up the debug UART connection and view diagnostic messages, connect the micro-USB cable from the micro-USB port on the RB3 Gen 2 device to the Linux host.



1. Install Minicom on the Linux host:

   ```
   sudo apt update
   sudo apt install minicom
   ```

2. Check the USB port:

   ```
   ls /dev/ttyUSB*
   ```

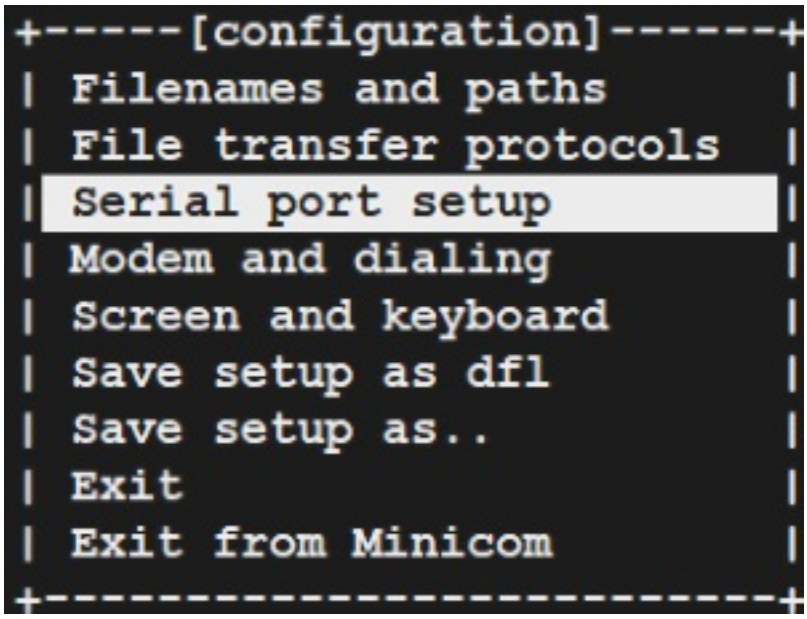
   Sample output

   ```
   /dev/ttyUSB0
   ```

3. Open Minicom:

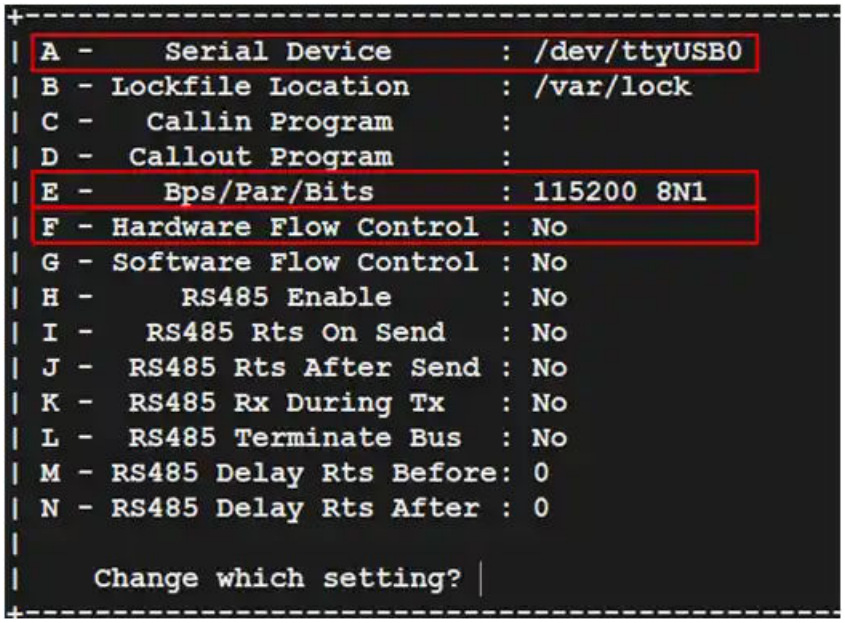
   ```
   sudo minicom -s
   ```

4. Press the Down arrow key to select the **Serial port setup** option. Use the Up and Down arrow keys to navigate through the menu.



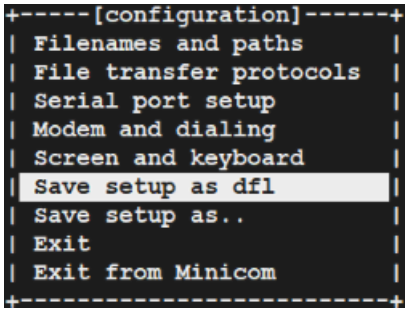
5. Set up the serial device configuration:
   a. Press **A** on your keyboard to set up the serial device name such as `/dev/ttyUSB0`.
   b. Press **Enter** to save the changes.
   c. Press **E** on your keyboard to set the baud rate. If the baud rate is not set to **115200**, press the **E** key again.
   d. Press **Q** on your keyboard to set the configuration to **8N1**.
   e. Press **Enter** to save the changes.
   f. Press **F** on your keyboard to set the **Hardware Flow Control** to `No`.

      NOTE Ensure that the letters A, E, Q, and F are in uppercase.



g. Press **Enter** to save the changes.

6. Select the **Save setup as dfl** option and press **Enter**.



7. Select **Exit** to open the UART console.
8. Log in to the UART console:
   - Login: `root`
   - Password: `oelinux123`

NOTE If the login console does not display as expected, verify the USB connection. If necessary, disconnect and then reconnect the micro-USB.

NOTE If you want to run sample applications from the UART shell, ensure to run applications in the Permissive mode:

```
setenforce 0
mount -o rw,remount /
```

How to SSH?

Set up Wi-Fi

Wi-Fi is operational in Station mode. The Wi-Fi host driver and the authentication for network management are initialized during the device boot up.

To update the Wi-Fi configuration, perform the following from the debug How to connect to a UART shell?:

1. Remount and enable read-and-write access to the default read-only `rootfs` file before editing the `/etc/wpa_supplicant.conf` file:

```
setenforce 0
mount -o rw,remount /
```

2. Stop the `wpa_supplicant`:

```
killall wpa_supplicant
```

3. Modify the content of the default `wpa_supplicant.conf` file to match the SSID and password of your router and open it using the on-device VI editor:

```
vi /etc/wpa_supplicant.conf
```

NOTE You can see the following configurations for security types specified in the default `wpa_supplicant.conf` file at `/etc` to add your required router configurations.

```
# Only WPA-PSK is used. Any valid cipher combination is accepted.
ctrl_interface=/var/run/sockets

network={
#Open
#       ssid="example open network"
#       key_mgmt=NONE
#WPA-PSK
#  Update the SSID to match that of the Wi-Fi SSID of your router.
ssid="QSoftAP"
#       proto=WPA RSN
#       key_mgmt=WPA-PSK
#       pairwise=TKIP CCMP
#       group=TKIP CCMP
# Update the password to match that of the Wi-Fi password of your router.
psk="1234567890"
#WEP
#       ssid="example wep network"
#       key_mgmt=NONE
#       wep_key0="abcde"
#       wep_key1=0102030405
#       wep_tx_keyidx=0
}
```

4. Save the modified `wpa_supplicant.conf` file and verify its content:

```
cat /etc/wpa_supplicant.conf
```

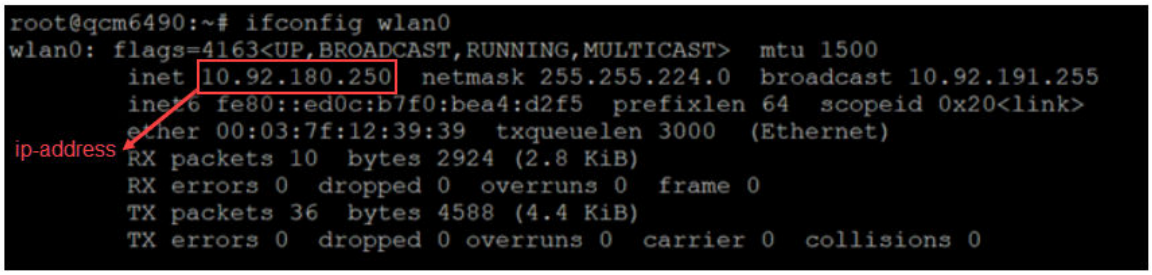
5. Reboot or power cycle the device. Wait for approximately one minute to establish a WLAN connection with the updated SSID and password.

6. **(Optional)** If you prefer not to reboot the device, run the following commands:

```
wpa_supplicant -Dnl80211 -iwlan0 -ddd -c /etc/wpa_supplicant.conf -f /tmp/
wpa_supplicant-log.txt &
dhcpcd wlan0
```

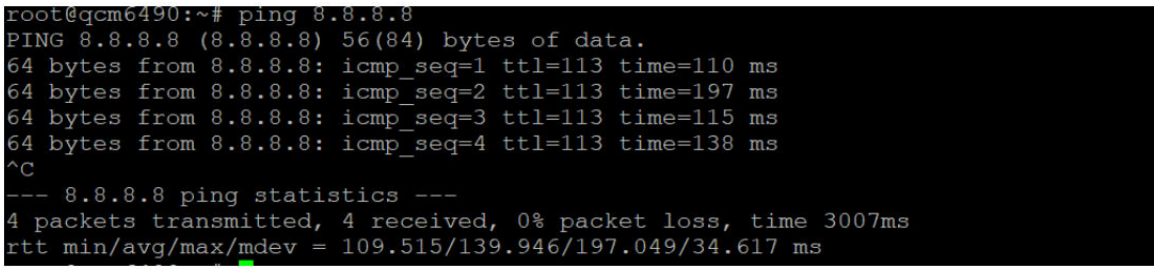
7. Check the WLAN connection status and IP address:

```
ifconfig wlan0
```



8. Ping the router to confirm the connection:

```
ping 8.8.8.8
```



Connect to SSH

NOTE Ensure that a Wi-Fi connection is established before connecting to SSH.

1. Find the IP address of the RB3 Gen 2 device in UART console:

```
ifconfig wlan0
```

2. Use the IP address obtained from **step 1** to SSH the device:

```
ssh root@ip-address
```

Example

```
ssh root@10.92.180.250
```

3. Connect to the SSH shell using the following password:

```
oelinux123
```

NOTE Ensure that the Linux host is connected to the same Wi-Fi access point.

NOTE To transfer the files successfully using the `scp` command, use the password `oelinux123`.

How to configure Ethernet with RJ45 port?

Ethernet/RJ45 port is enabled as a downstream port of PCIe to USB controller (`renesas`). Ensure that `renesas_usb_fw.mem` is available at the `/lib/firmware` directory.

NOTE If `renesas_usb_fw.mem` firmware is not available at the `/lib/firmware` directory, then How to connect to a UART shell? and enable Wi-Fi.

After getting SSH and IP address, How to update USB and Ethernet controller firmware?.

To check if USB to ETH controller is enumerated, run the following command:

```
lsusb
```

Sample output:

```
Bus 002 Device 003: ID 0b95:1790 ASIX Electronics Corp. AX88179 Gigabit
Ethernet
Bus 002 Device 002: ID 05e3:0625 Genesys Logic, Inc. USB3.2 Hub
Bus 002 Device 001: ID 1d6b:0003 Linux Foundation 3.0 root hub
Bus 001 Device 002: ID 05e3:0610 Genesys Logic, Inc. Hub
Bus 001 Device 001: ID 1d6b:0002 Linux Foundation 2.0 root hub
```

Connect an RJ45 cable to the RB3 Gen 2 device.

To check the Ethernet IP address, run the following command:

```
ifconfig
```

Sample output:

NOTE 10.219.0.106 is the IP address.

```
enP1p4s0u1u1 Link encap:Ethernet HWaddr A6:CD:9B:FD:C1:B5
          inet addr:10.219.0.106  Bcast:10.219.1.255  Mask:255.255.254.0
          inet6 addr: fe80::a370:7a00:8131:5a03/64 Scope:Link
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1071 errors:0 dropped:0 overruns:0 frame:0
          TX packets:132 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:60711 (59.2 KiB)  TX bytes:18342 (17.9 KiB)
```

How to update USB and Ethernet controller firmware?

If you encounter any USB or Ethernet connectivity issues on the RB3 Gen 2 device, a firmware update for the PCIe to USB controller could be a solution.

Prerequisite: The device should be connected to the SSH terminal.

1. To download the firmware, you must register and log in with your ID.
2. Transfer the files using SCP to the following path: `/lib/firmware`.

   NOTE The firmware name should be `renesas_usb_fw.mem` and if not, you must rename it. The driver identifies the firmware based on this specific filename, and other firmware names will not function.

Example:

```
scp renesas_usb.mem root@<ip address>:/lib/firmware
```

NOTE When prompted for a password, enter `oelinux123`.

NOTE After flashing the firmware, reboot the device. Upon subsequent boot-up, the driver reads the firmware and activates the PCIe-to-USB controller.

11 References

11.1 Workspace view

This section provides sample workspace structures with `qsc-cli` and GitHub workflow standalone use cases for QCS6490 and QCS5430.

Workspace structure with qsc-cli

- Directory structure before the `Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK` distribution build is shown in the following figure (`LE.QCLINUX.1.0.r1` contains the Yocto workspace):

```
├── about.html
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY22
│   │   ├── snap_release.xml
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-hwe
│   │   ├── meta-qcom-qim-product-sdk
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       └── snap_release.xml
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa.gsifw
│   │   └── sectoolsv2
│   └── contents.xml
```

- Directory structure after `Qualcomm_Linux.SPF.1.0|TEST|DEVICE|PB_QIMPSDK` distribution build is shown in the following figure:

```
├── about.html
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY22
│   │   ├── snap_release.xml
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── build-qcom-wayland
│   │   ├── bitbake-cookerdaemon.log
│   │   ├── bitbake.lock
│   │   ├── bitbake.sock
│   │   ├── buildhistory
│   │   ├── cache
│   │   ├── conf
│   │   ├── qim-prod-sdk
│   │   ├── qim-sdk
│   │   ├── tflite-sdk
│   │   └── tmp-glibc
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-hwe
│   │   ├── meta-qcom-qim-product-sdk
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       └── snap_release.xml
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa.gsifw
│   │   └── sectoolsv2
│   └── contents.xml
```

- Directory structure before `Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem` distribution build with firmware and extras is shown in the following figure:

```
├── about.html
├── ADSP.HT.5.5.c8
│   └── adsp_proc
├── AOP.HO.3.6
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
├── BTFW.HSP.2.1.2
│   └── btfw_proc
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
├── CDSP.HT.2.5.c3
│   └── cdsp_proc
├── CPUCP.FW.1.0
│   └── cpucp_proc
├── DSP.AT.1.0
│   └── dsp_proc
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   ├── layers
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   └── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
├── QCM6490.LE.1.0
│   ├── common
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
├── TZ.APPS.1.0
│   ├── qtee_tas
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   └── trustzone_images
├── WLAN.HSP.1.1
│   └── wlan_proc
└── WLAN.MSL.2.0.c2
    └── wlan_proc
```

- Directory structure after `Qualcomm_Linux.SPF.1.0|AP|Standard|OEM|NoModem` distribution build with firmware and extras is shown in the following figure:

```
├── about.html
├── ADSP.HT.5.5.c8
│   ├── adsp_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_kodiak.adsp.prodQ.json
├── AOP.HO.3.0.2
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── AOP.HO.3.6
│   ├── aop_proc
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
├── BTFW.HSP.2.1.2
│   └── btfw_proc
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
├── CDSP.HT.2.5.c3
│   ├── BuildProducts.txt
│   ├── cdsp_proc
│   └── VariantImgInfo_kodiak.cdsp.prodQ.json
├── CPUCP.FW.1.0
│   └── cpucp_proc
├── DSP.AT.1.0
│   └── dsp_proc
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   ├── build-qcom-wayland
│   ├── downloads
│   ├── layers
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   ├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
│   └── sstate-cache
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   └── ufs
│   │       └── bin
│   │           ├── QCM6490_bootbinaries.zip
│   │           ├── QCM6490_dspso.zip
│   │           └── QCM6490_fw.zip
│   └── contents.xml
├── QCS9100.LE.1.0
│   ├── common
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
├── TZ.APPS.1.0
│   ├── qtee_tas
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   ├── BuildProducts.txt
│   ├── HY22
│   └── trustzone_images
├── WLAN.HSP.1.1
│   └── wlan_proc
└── WLAN.MSL.2.0.c2
    └── wlan_proc
```

Workspace structure with GitHub workflow standalone instructions

- Directory structure before GitHub workflow QIMP SDK standalone build is shown in the following figure:

```
├── layers
│   ├── bitbake-cookerdaemon.log
│   ├── meta-openembedded
│   ├── meta-qcom
│   ├── meta-qcom-distro
│   ├── meta-qcom-hwe
│   ├── meta-qcom-qim-product-sdk
│   ├── meta-rust
│   ├── meta-security
│   ├── meta-selinux
│   ├── meta-virtualization
│   └── poky
└── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
```

- Directory structure after GitHub workflow QIMP SDK standalone build is shown in the following figure:

```
├── build-qcom-wayland
│   ├── bitbake-cookerdaemon.log
│   ├── buildhistory
│   ├── cache
│   ├── conf
│   ├── qim-prod-sdk
│   ├── qim-sdk
│   ├── tflite-sdk
│   └── tmp-glibc
├── downloads
│   ├── a52dec-0.7.4.tar.gz
│   ├── a52dec-0.7.4.tar.gz.done
│   ├── acl-2.3.1.tar.gz
      --<ALL downloads Contents>
│   └── zlib-1.2.11.tar.xz.done
├── layers
│   ├── bitbake-cookerdaemon.log
│   ├── meta-openembedded
│   ├── meta-qcom
│   ├── meta-qcom-distro
│   ├── meta-qcom-hwe
│   ├── meta-qcom-qim-product-sdk
│   ├── meta-rust
│   ├── meta-security
│   ├── meta-selinux
│   ├── meta-virtualization
│   └── poky
├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
└── sstate-cache
    ├── 00
    ├── 01
     --<sstate-cache-objects>
```

- Directory structure after building firmware of `qualcomm-linux-spf-1-0_ap_standard_oem_nomodem` is shown in the following figure:

NOTE

- `qualcomm-linux-spf-1-0_ap_standard_oem_nomodem` contains the downloaded select firmware sources.
- `LE.QCLINUX.1.0.r1` contains the built Yocto workspace.

```
├── about.html
├── ADSP.HT.5.5.c8
│   ├── adsp_proc
│   │   ├── avs
│   │   ├── build
│   │   ├── config
│   │   ├── core
│   │   ├── crashman
│   │   ├── dsp_kodiak.adsp.prodQ.elf
│   │   ├── eai_lib
│   │   ├── gpr
│   │   ├── image_info.json
│   │   ├── loc
│   │   ├── performance
│   │   ├── platform
│   │   ├── qdsp6
│   │   ├── qmimsgs
│   │   ├── qsh_algorithms
│   │   ├── qsh_api
│   │   ├── qsh_platform
│   │   ├── ssc_algorithms
│   │   ├── ssc_drivers
│   │   └── tools
│   ├── BuildProducts.txt
│   └── VariantImgInfo_kodiak.adsp.prodQ.json
├── AOP.HO.3.6
│   ├── aop_proc
│   │   ├── build
│   │   ├── core
│   │   ├── pack
│   │   └── tools
│   ├── BuildProducts.txt
│   └── VariantImgInfo_AAAAANAZO.json
├── BOOT.MXF.1.0.c1
│   └── boot_images
│       ├── boot
│       ├── boot_tools
│       ├── Build
│       ├── BuildLogs
│       ├── edk2
│       ├── sdk
│       ├── sectools
│       └── ssg
├── BTFW.HSP.2.1.2
│   └── btfw_proc
│       ├── btfn
│       ├── build
│       └── out
├── BTFW.MOSELLE.1.1.0
│   └── btfw_proc
│       ├── btfn
│       ├── build
│       └── out
├── CDSP.HT.2.5.c3
│   ├── BuildProducts.txt
│   ├── cdsp_proc
│   │   ├── build
│   │   ├── ceng
│   │   ├── config
│   │   ├── core
│   │   ├── crashman
│   │   ├── cvp
│   │   ├── dsp_kodiak.cdsp.prodQ.elf
│   │   ├── image_info.json
│   │   ├── performance
│   │   ├── platform
│   │   ├── qdsp6
│   │   ├── qmimsgs
│   │   ├── tools
│   │   ├── vap
│   │   └── video_processing
│   └── VariantImgInfo_kodiak.cdsp.prodQ.json
├── CPUCP.FW.1.0
│   └── cpucp_proc
│       ├── kodiak
│       └── lemansau
├── DSP.AT.1.0
│   └── dsp_proc
│       ├── ams_spf
│       ├── avs
│       ├── awe
│       ├── build
│       ├── ceng
│       ├── config
│       ├── core
│       ├── crashman
│       ├── data
│       ├── eai_lib
│       ├── gpr
│       ├── image_info.json
│       ├── performance
│       ├── platform
│       ├── qdsp6
│       ├── qmimsgs
│       └── tools
├── LE.QCLINUX.1.0.r1
│   ├── apps_proc
│   │   ├── build_levm.sh
│   │   ├── prebuilt_HY11
│   │   ├── snap_release.xml
│   │   ├── sources
│   │   ├── syncbuild.sh
│   │   └── sync_snap_v2.sh
│   ├── build-qcom-wayland
│   │   ├── bitbake-cookerdaemon.log
│   │   ├── buildhistory
│   │   ├── cache
│   │   ├── conf
│   │   └── tmp-glibc
│   ├── downloads
│   │   ├── acl-2.3.1.tar.gz
│   │   ├── acl-2.3.1.tar.gz.done
│   │   │   ···<ALL downloads Contents>
│   ├── layers
│   │   ├── meta-openembedded
│   │   ├── meta-qcom
│   │   ├── meta-qcom-distro
│   │   ├── meta-qcom-extras
│   │   ├── meta-qcom-hwe
│   │   ├── meta-rust
│   │   ├── meta-security
│   │   ├── meta-selinux
│   │   ├── meta-virtualization
│   │   └── poky
│   ├── qualcomm-linux-spf-1-0_hlos_oem_metadata
│   │   ├── about.html
│   │   ├── QCM6490.LE.1.0
│   │   ├── QCM8550.LE.2.0
│   │   └── QCS9100.LE.1.0
│   ├── setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
│   └── sstate-cache
│       ├── 00
│       ├── 01
│       │   ···<sstate-cache-objects>
├── LKP.QCLINUX.1.0.r1
│   └── kernel_platform
│       ├── qcom
│       ├── snap_release.xml
│       └── sync_snap_v2.sh
├── QCM6490.LE.1.0
│   ├── common
│   │   ├── build
│   │   ├── config
│   │   ├── core_qupv3fw
│   │   ├── dataipa_gsifw
│   │   └── sectoolsv2
│   └── contents.xml
├── QCS9100.LE.1.0
│   ├── common
│   │   ├── aurixfw
│   │   ├── build
│   │   ├── cdt
│   │   ├── config
│   │   ├── Core
│   │   └── sectoolsv2
│   └── contents.xml
├── SAIL.SI.1.0
│   └── sail_proc
│       ├── BSP
│       └── build
├── TZ.APPS.1.0
│   ├── qtee_tas
│   │   ├── apps
│   │   ├── build
│   │   └── sdk
│   ├── ta_size_info_kodiak.csv
│   └── ta_size_info_lemans.csv
├── TZ.XF.5.0
│   ├── BuildProducts.txt
│   ├── HY22
│   │   └── trustzone_images
│   └── trustzone_images
│       ├── build
│       ├── core
│       ├── scons_dep_tree_EACAANAA_2eb56721f.txt
│       ├── sectools
│       ├── signed_images.json
│       ├── ssg
│       └── tools
├── WLAN.HSP.1.1
│   └── wlan_proc
│       ├── build
│       ├── config
│       ├── core
│       ├── scripts
│       ├── tools
│       └── wlan
└── WLAN.MSL.2.0.c2
    └── wlan_proc
        ├── build
        ├── scripts
        ├── tools
        └── wlan
```

- Directory structure after building firmware of `qualcomm-linux-spf-1-0_amss_standard_oem_nomodem` is shown in the following figure:

  NOTE
  - `qualcomm-linux-spf-1-0_amss_standard_oem_nomodem` contains the downloaded select firmware sources.
  - `LE.QCLINUX.1.0.r1` contains the built Yocto workspace.


about.html
ADSP.HT.5.5.c8
adsp_proc
avs
build
config
core
crashman
dsp_kodiak.adsp.prodQ.elf
eai_lib
gpr
image_info.json
loc
performance
platform
qdsp6
qmimsgs
qsh_algorithms
qsh_api
qsh_platform
ssc_algorithms
ssc_drivers
tools
BuildProducts.txt
VariantImgInfo_kodiak.adsp.prodQ.json
ADSP.HT.5.8.c4
adsp_proc
avs
build
config
core
crashman
eai_lib
enpu
gpr
image_info.json
loc
performance
platform
qdsp6
qmimsgs
qsh
sdc_core
sdc_sensors
ssc
ssc_api
ssc_drivers
tools
ADP.HO.3.6
aop_proc
build
core
pack
tools
BuildProducts.txt
VariantImgInfo_AAAAANAZO.json
BOOT.MXF.1.0.c1
boot_images
boot
boot_tools
Build
BuildLogs
edk2
sdk
sectools
ssg
BTFW.HSP.2.1.2
btfw_proc
btfm
build
out
BTFW.MOSELLE.1.1.0
btfw_proc
btfm
build
out
CDSP.HT.2.5.c3
BuildProducts.txt
cdsp_proc
build
ceng
config
core
crashman
cvp
dsp_kodiak.cdsp.prodQ.elf
image_info.json
performance
platform
qdsp6
qmimsgs
tools
vap
video_processing
VariantImgInfo_kodiak.cdsp.prodQ.json
CPUCP.FW.1.0
cpucp_proc
kodiak
lemansau
LE.QCLINUX.1.0.r1
apps_proc
build_levm.sh
prebuilt_HY11
snap_release.xml
sources
syncbuild.sh
sync_snap_v2.sh
build-qcom-wayland
bitbake-cookerdaemon.log
bitbake.lock
bitbake.sock
buildhistory
cache
conf
tmp-glibc
downloads
acl-2.3.1.tar.gz
acl-2.3.1.tar.gz.done
<--ALL downloads Contents>
layers
meta-openembedded
meta-qcom
meta-qcom-distro
meta-qcom-extras
meta-qcom-hwe
meta-rust
meta-security
meta-selinux
meta-virtualization
poky
qualcomm-linux-spf-1-0_hlos_oem_metadata
about.html
QCM6490.LE.1.0
QCM8550.LE.2.0
QCS9100.LE.1.0
setup-environment -> layers/meta-qcom-distro/set_bb_env.sh
sstate-cache
00
01
<--sstate-cache-objects>
LKP.QCLINUX.1.0.r1
kernel_platform
qcom
snap_release.xml
sync_snap_v2.sh
MPSS.HI.4.3.3.c6.2
modem_proc
build
datamodem
mcfg
rf
QCM6490.LE.1.0
common
build
config
core_qupv3fw
dataipa_gsifw
sectoolsv2
contents.xml
SAIL.SI.1.0
sail_proc
BSP
build
TZ.APPS.1.0
qtee_tas
apps
build
sdk
ta_size_info_kodiak.csv
ta_size_info_lemans.csv
TZ.XF.5.0
BuildProducts.txt
HY22
trustzone_images
trustzone_images
build
core
scons_dep_tree_EACAANAA_b11b39829.txt
sectools
signed_images.json
ssg
tools
WLAN.HSP.1.1
wlan_proc
build
config
core
scripts
tools
wlan
WLAN.MSL.2.0.c2
wlan_proc
build
scripts
tools
wlan

Images directory structure after successful build

- Images directory after a successful build is shown in the following figure:

```
├── aop.mbn
├── cpucp.elf
├── devcfg.mbn
├── dtb.bin
├── efi.bin
├── gpt_backup0.bin
├── gpt_backup1.bin
├── gpt_backup2.bin
├── gpt_backup3.bin
├── gpt_backup4.bin
├── gpt_backup5.bin
├── gpt_main0.bin
├── gpt_main1.bin
├── gpt_main2.bin
├── gpt_main3.bin
├── gpt_main4.bin
├── gpt_main5.bin
├── hypvm.mbn
├── Image
├── imagefv.elf
├── kernel-modules.tgz
├── multi_image.mbn
├── patch0.xml
├── patch1.xml
├── patch2.xml
├── patch3.xml
├── patch4.xml
├── patch5.xml
├── prog_firehose_ddr.elf
├── prog_firehose_lite.elf
├── qdl
├── qupv3fw.elf
├── rawprogram0.xml
├── rawprogram1.xml
├── rawprogram2.xml
├── rawprogram3.xml
├── rawprogram4.xml
├── rawprogram5.xml
├── shrm.elf
├── system.img
├── tools.fv
├── tz.mbn
├── uefi.elf
├── uefi_sec.mbn
├── vmlinux
├── xbl_config.elf
├── xbl.elf
├── XblRamdump.elf
└── zeros_5sectors.bin
```

The images directory files are described in the following table:

Filename	Description
`.mbn` and `*.elf`	Boot critical images
`gpt_main*.bin`	GUID partition table binaries for the primary partition table
`gpt_backup*.bin`	GUID partition table binaries for the secondary partition table
`system.img`	Rootfs image
`rawprogram*.xml`	Image `lun` and start sector `lba` values
`efi.bin`	EFI system partition image. For more information, see Qualcomm Linux Yocto Guide.
`qdl`	Flashing tool binary
`dtb.bin`	Binary image that bundles all DTB binaries generated during build
`vmlinux`	Compile kernel `elf` binary
`Image`	Linux kernel ARM64 boot executable image

11.2 Related documents

Document title	Document number
Qualcomm Technologies, Inc.	
Qualcomm Linux Yocto Guide	80-70014-27
Qualcomm Linux Release Notes	RNO-240626095531
Qualcomm Linux Kernel Guide	80-70014-3
RB3 Gen 2 Quick Start Guide	80-70014-253
Qualcomm Intelligent Multimedia Product (QIMP) SDK Quick Start Guide	80-70014-51
Qualcomm® Linux Intelligent Robotics Product SDK (QIRP SDK) 2.0 Quick Start Guide	80-65220-2
Qualcomm Linux Virtual Machine Setup Guide	80-70014-41
Qualcomm Software Center User Guide	80-72780-2
Resources	
Qualcomm manifest	https://github.com/quic-yocto/qcom-manifest
Yocto central download directory	https://docs.yoctoproject.org/4.0.16/singleindex.html#term-DL_DIR
CodeLinaro patch file	https://artifacts.codelinaro.org/artifactory/codelinaro-le/0001-fetch2-git-Add-verbose-logging-support.patch
meta-qcom-realtime	https://github.com/quic-yocto/meta-qcom-realtime

11.3 Acronyms and terms

Acronym or term	Definition
aDSP	Advanced digital signal processor
cDSP	Compute digital signal processor

Acronym or term	Definition
ADB	Android debug bridge
AOP	Always on processor
BSP	Board support package
CDT	Configuration data table
CLI	Command-line interface
EDL	Emergency download
EFI	Extensible firmware interface
LE	Linux embedded
PCAT	Product configuration assistant tool
QDL	Qualcomm download
QIMP	Qualcomm Intelligent Multimedia Product
QIRF	Qualcomm Intelligent Robotics Function
QIRP	Qualcomm Intelligent Robotics Product
QLI	Qualcomm Linux
QNN	Qualcomm neural network
QPM	Qualcomm Package Manager
QSC	Qualcomm software center
QUD	Qualcomm USB drivers
SDK	Software development kit
TZ	TrustZone

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.