Qualcomm Technologies, Inc.

Qualcomm® Intelligent Multimedia SDK (QIM SDK) Quick Start Guide

80-50450-51 Rev. AD

October 10, 2023

Revision history

Revision	Date	Description
AA	August 8, 2023	Initial release
AB	August 9, 2023	This revision has been rolled out to adhere to QTI standard documentation practices, no technical content was changed.
AC	September 2023	Updated the document for QIM SDK v1.0.1
AD	October 2023	■ Sync and build QIM SDK with container: □ Updated the JSON configuration in Step 4 □ Added procedures to enable AI inference plug-ins for Qualcomm Neural Processing SDK ■ Added Sync and build QIM SDK with Linux workstation ■ Added Troubleshoot host setup

Contents

Figures

Tables

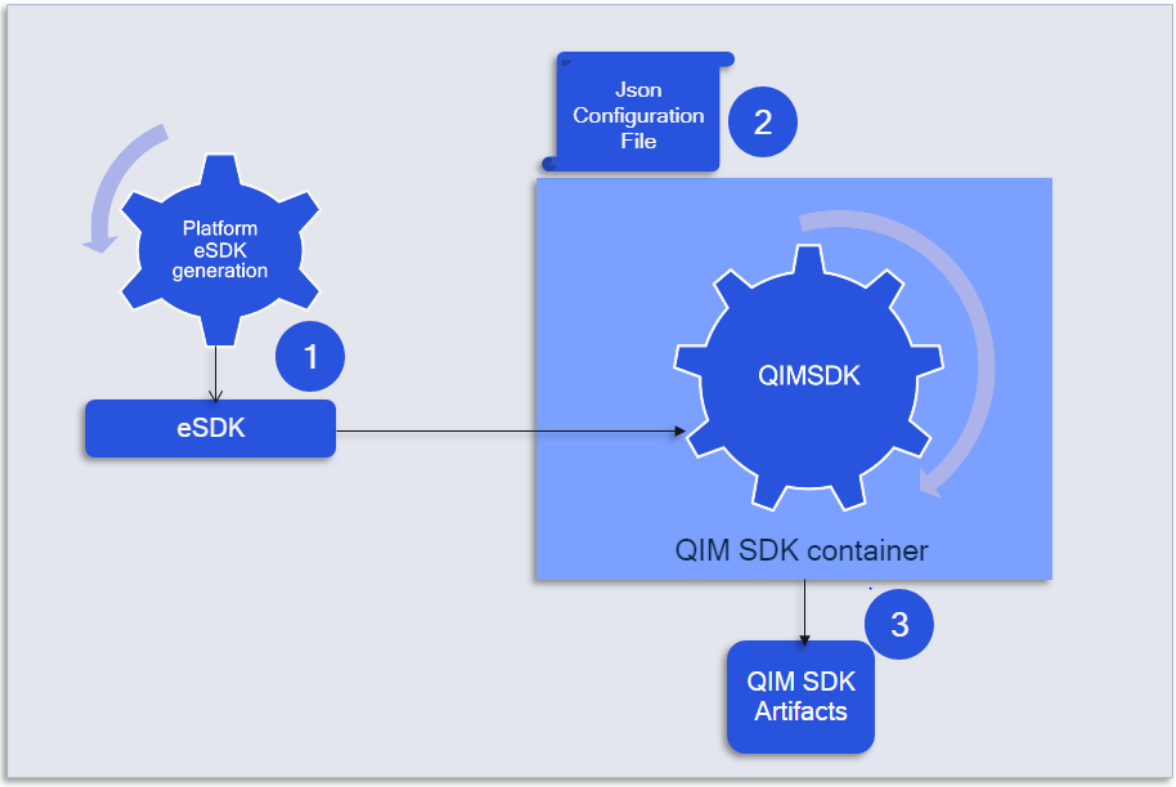
1 Introduction

The Qualcomm intelligent multimedia software development kit (QIM SDK) is a GStreamer-based SDK that provides a set of multimedia, computer vision (CV), and artificial intelligence (AI) plug-ins to facilitate application developers to develop suitable applications.

This document provides step-by-step instructions to compile a standalone QIM SDK and set up the development environment. It describes the steps for enabling the decouple developer workflow, which includes setting up the build environment where the application developer can compile the QIM SDK and can also develop and compile the applications.

For assistance or clarification on information in this document, see https://www.qualcomm.com/support.

The following figure provides a summary of the decouple workflow:



The tool requires a platform extensible SDK (eSDK) and a configuration file (JSON format) to generate the QIM SDK artifacts.

The table shows QIM SDK version mapping with CodeLinaro release tag:

Table 1-1 Release information

QIM SDK version	CodeLinaro release tag
V1.0.0	■ QIM.SDK.1.0.0.r1-00600-QIM.0 ■ QIM.SDK.1.0.0.r1-01400-QIM.0
V1.0.1	QIM.SDK.1.0.0.r1-01100-QIM.0

References

Table 1-2 Related documents

Title	Number
Qualcomm	
Qualcomm Intelligent Multimedia SDK (QIM SDK) Reference	80-50450-50
Qualcomm TensorFlow Lite Software Development Kit Tools Quick Start Guide	80-50450-52
Release Note for QCS8550.LE.1.0 for v1.0.0	RNO-230630011851
Release Note for QCS8550.LE.1.0 for v1.0.1	RNO-230830225415
Resources	
https://source.android.com/docs/setup/start/initializing	–

Table 1-3 Acronyms and definitions

Acronym or term	Definition
AI	Artificial intelligence
BIOS	Basic input/output system
CV	Computer vision
eSDK	Extensible software development kit
QIM SDK	Qualcomm intelligent multimedia software development kit

2 Set up QIM SDK build environment

The QIM SDK is released in source form; therefore, establishing the build environment to compile it is a mandatory but one-time setup.

Prerequisites:

- Ensure that you have `sudo` access to the Linux host machine.
- Ensure that the Linux host version is Ubuntu 18.04 or Ubuntu 20.04.
- Increase the maximum user watches and maximum user instances on the host system.
- Add the following command lines to `/etc/sysctl.conf` and reboot the host:

```
fs.inotify.max_user_instances=8192
fs.inotify.max_user_watches=542288
```

2.1 Install required host packages

Run the commands to install the host packages on the Linux host machine:

```
$ sudo apt install -y jq
$ sudo apt install -y texinfo chrpath libxml-simple-perl openjdk-8-jdk-
headless
```

For Ubuntu 18.04 and higher:

```
$ sudo apt-get install git-core gnupg flex bison build-essential zip curl
zlib1g-dev gcc-multilib g++-multilib libc6-dev-i386 libncurses5 lib32ncurses5-
dev x11proto-core-dev libx11-dev lib32z1-dev libgl1-mesa-dev libxml2-utils
xsltproc unzip fontconfig
```

For more information, seehttps://source.android.com/docs/setup/start/initializing.

2.2 Set up docker environment

A docker is a platform used to build, develop, test, and deliver software. To compile the QIM SDK, the docker must be configured on the Linux host machine.

Ensure that CPU virtualization is enabled on the Linux host machine. If it is not enabled, do the following to enable it from the basic input/output system (BIOS) configuration settings:

1. Enable virtualization from BIOS:
   a. Press **F1** or **F2** when the system is booting up to step into BIOS. The BIOS window is displayed.
   b. Switch to the **Advanced** tab.
   c. In the **CPU Configuration** section, set **Virtualization Technology** to **Enabled**.
   d. Press **F12** to save and exit, and then restart the system.

   If these steps do not work, follow the specific instructions from the system provider to enable the virtualization.
2. Remove any old instances of the docker:

```
$ sudo apt remove docker-desktop
$ rm -r $HOME/.docker/desktop
$ sudo rm /usr/local/bin/com.docker.cli
$ sudo apt purge docker-desktop
```

3. Set up the docker remote repository:

```
$ sudo apt-get update
$ sudo apt-get install ca-certificates curl gnupg lsb-release
$ sudo mkdir -p /etc/apt/keyrings
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --
dearmor -o /etc/apt/keyrings/docker.gpg
$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/
docker.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs)
stable" |  sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

4. Install docker engine:

```
$ sudo apt-get update
$ sudo apt-get install docker-ce docker-ce-cli
```

5. Add user to docker group:

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

6. Reboot the system.

3 Generate platform eSDK

The platform eSDK is a mandatory requirement to compile the QIM SDK. It provides all the required platform dependencies required by the QIM SDK.

Do the following to generate the platform eSDK:

1. Create a build for the QCS8550.LE.1.0 release. The instructions to create a build are available in the release notes. To access the release notes, see References.

   NOTE If the images were previously built, execute steps 2 and 3, and then create a clean build.

2. Add the following line in the `qti-robotics-image.bb` recipe file.

   ```
   <Workspace>/poky/meta-qti-bsp/recipes-products/images/qti-robotics-image.bb
   CORE_IMAGE_EXTRA_INSTALL:remove:kalama = "packagegroup-qti-gst"
   ```

3. Run the following command to build the user space images and platform eSDK:

   For QCS8550.LE.1.0:

   ```
   $ bitbake -fc populate_sdk_ext qti-robotics-image
   ```

4 Build QIM SDK – Developer workflow

The QIM SDK workflow requires the developer to provide the configuration file with valid input entries. The helper shell scripts from the sdk-tools project (present in the QIM SDK source tree) provide helper utility functions to set up shell environment, which the developer can use for the QIM SDK workflow.

The developer builds the QIM SDK projects within the container and generates the artifacts using the utilities provided by sdk-tools.

After a QIM SDK container is built, the developer can attach to the container and use the helper utilities in the container shell environment for continuous development.

- There is a provision to install the QIM SDK artifacts to a Qualcomm device connected to the Linux host via USB/adb.
- There is also a provision to copy the QIM SDK artifacts from the container to a different host machine where the Qualcomm device is connected.

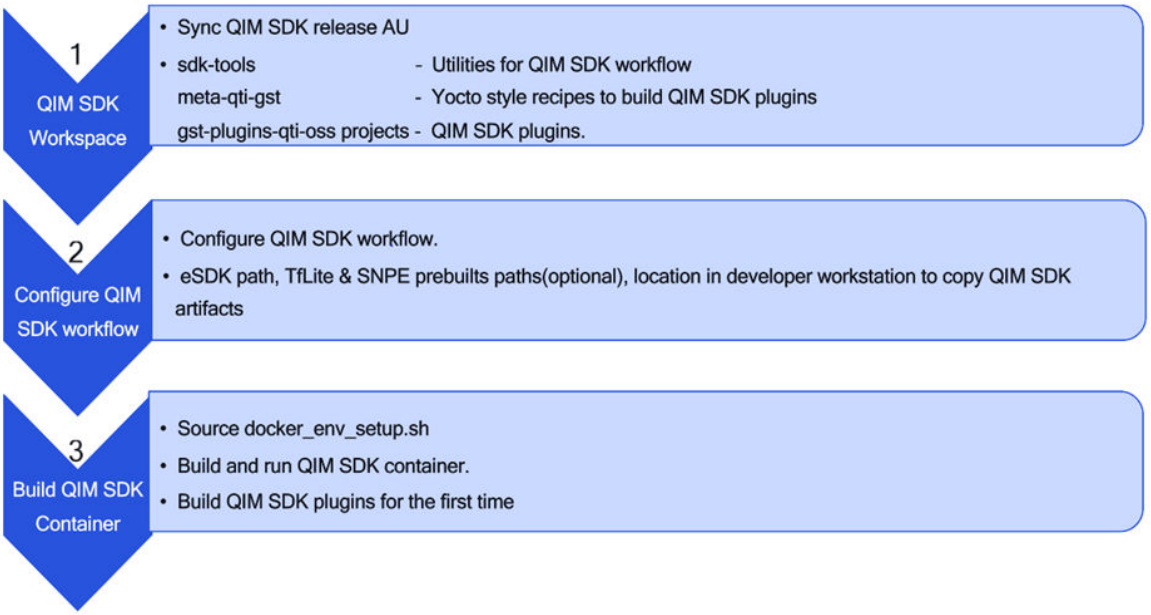


Figure 4-1 QIM SDK container workflow

The following figure lists the set of utilities available after setting up the container environment using the helper scripts for building QIM SDK.

```
Docker build environment setup
==============================
Docker environment setup ready !!!

qimsdk-docker-build-image <path-to-config-json>
    Build docker image based on Dockerfile in /local/mnt/workspace/divyathe/QIM.SDK.1.0.0/sdk-tools
qimsdk-docker-run-container <path-to-config-json>
    Run docker container based on compiled docker image
qimsdk-docker-rm-container <path-to-config-json>
    Remove docker container based on compiled docker image
qimsdk-docker-start-container <path-to-config-json>
    Start docker container based on compiled docker image
qimsdk-docker-stop-container <path-to-config-json>
    Stop docker container based on compiled docker image
qimsdk-docker-build-and-sync-artifacts-all <path-to-config-json>
    Build docker image and extract QIMSDK Artifacts to user specified directory in host machine
qimsdk-docker-build-and-sync-artifacts-rel <path-to-config-json>
    Build docker image and extract QIMSDK Release Artifacts to user specified directory in host machine
qimsdk-docker-build-and-sync-artifacts-dev <path-to-config-json>
    Build docker image and extract QIMSDK Development Artifacts to user specified directory in host machine
qimsdk-docker-cleanup
    Clean up old docker images and build cache
```

Figure 4-2 Available utilities after sourcing the container environment

The following figure shows the sequence of execution of the utilities:

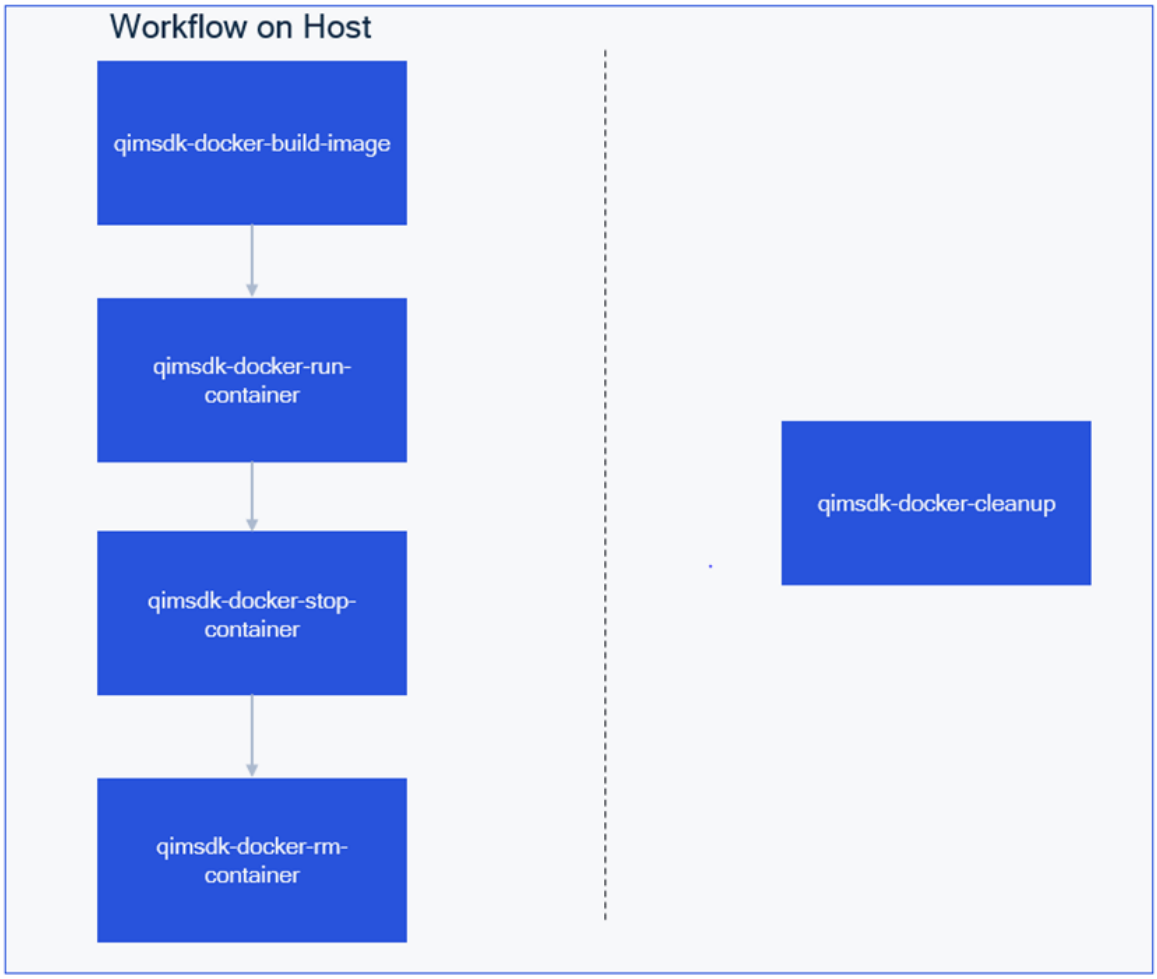


Figure 4-3 Sequence of utilities on host

4.1 Sync and build QIM SDK with container

QIM SDK is compiled when the docker image is created.

To sync and build QIM SDK, do the following:

1. Create a directory on the host file system to sync the QIM SDK workspace. For example:

   ```
   $mkdir <qim-sdk-workspace>
   $cd <qim-sdk-workspace>
   ```

2. Fetch the QIM SDK source code from CodeLinaro:

   ```
   $ repo init -u https://git.codelinaro.org/clo/le/sdkqim/qim/manifest.git --repo-branch=qc/stable --repo-url=git://git.quicinc.com/tools/repo.git -m QIM.SDK.1.0.0.r1-01100-QIM.0.xml -b release && repo sync -qc --no-tags -j8
   ```

3. Create a directory on the host file system that can be mounted onto the docker.

   For example: `mkdir -p <qim-sdk-workspace>/<host_dir>`

This directory can be created anywhere on the Linux host machine, and it does not depend on where the QIM SDK project is synced.

After the workflow is completed within the container, the QIM SDK artifacts can be found at the directory created in this step.

4. Edit the JSON configuration file present in `<qim-sdk-workspace>/sdk-tools/targets/LE.PRODUCT.2.1` with the following entries:

```
{
"Image_OS": "ubuntu18",
"Additional_tag": "sdk-18",
"eSDK_shell_file": "Absolute path to Platform extensible SDK file on Linux
Workstation",
"Base_Dir_Location": "path-to-the-directory-where-the-project-is-
initialized-applies-only-for-Linux-Workstation(host)-build-variant",

"Tflite_prebuilt_file": "absolute path to tf-lite-prebuilt-dev-tar-file
generated from TfLite SDK",
"Acceleration_engines":
[
{
"Acceleration_engine":"name of acceleration engine 1",
"Acceleration_engine_path":"absolute path to Acceleration Engine SDK 1"
},
{
"Acceleration_engine":"-", "Acceleration_engine_path":"-"
}
],
"Host_dir_mounted_in_container": "<Directory on Host that will be mapped
into QIMSDK container to get artifacts to host (not applicable to Linux
Workstation workflow)>",
"Deploy_URL": "/home/<container username>/work/sync_rel",
"Deploy_dev_URL": "/home/<container username>/work/sync_dev",
"Deploy_QIMSDK_Artifacts_URL":"Directory on Host where all QIMSDK
artifacts archive will be made available",
"Gst_plugins_qti_oss_dependencies":["cairo", "ffmpeg", "gdk-pixbuf",
"liba52", "libgudev", "lame", "librsvg", "libtheora", "libusb1",
"libwebp", "mpg123", "orc", "sbc", "speex", "taglib", "vulkan-loader"]
}
```

For more information on the entries mentioned in the JSON configuration file, see the `Docker.md` readme file present at `<qim-sdk-workspace>/sdk-tools/`.

Enable AI inference plug-ins:

Qualcomm Neural Processing SDK

QIM SDK workflow can generate the QIM SDK plugin for Qualcomm Neural Processing SDK (Formerly known as SNPE).

To build the SNPE QIM SDK plug-in, download the SNPE SDK needs to be downloaded and path to SNPE sdk on Linux workstation needs to be provided in configuration JSON file.

a. To download Qualcomm Package Manager 3 for Ubuntu, clickhttps://qpm.qualcomm.com/, and click **Tools**.

b. In the left pane, in the **Search Tools** field, type `QPM`. From the **System OS** list, select **Linux**.

   The search results display a list of Qualcomm Package Managers.

c. Select Qualcomm Package Manager 3 and download the Linux debian package.

d. To install Qualcomm Package Manager 3 for Linux, use the following command:

```
$ dpkg -i --force-overwrite /path/to/
QualcommPackageManager3.3.0.83.1.Linux-x86.deb
```

e. To download Qualcomm Snapdragon Neural Processing Engine SDK, clickhttps://qpm.qualcomm.com/, and click **Tools**.

f. In the left pane, in the **Search Tools** field, type `ai stack`. From the **System OS** list, select **Linux**.

   The search results display a list of Qualcomm Package Managers..

g. Click Qualcomm® Neural Processing SDK, and then download the .qik package with Linux version 2.13.0.

h. To install Qualcomm® Neural Processing SDK on Ubuntu workstation, run the following commands:

```
$ qpm-cli --login <username>
$ qpm-cli --license-activate qualcomm_neural_processing_sdk
$ qpm-cli --extract <full path to downloaded .qik file>
```

   After the extraction is successful, the following message is displayed."`/opt/qcom/aistack/snpe/2.13.4.230831`"

i. Add the entries into `LE.PRODUCT.2.1.json` file. For example:

```
"Acceleration_engines":
[
{"Acceleration_engine":"snpe",
"Acceleration_engine_path":”/opt/qcom/aistack/snpe/2.13.4.230831"
},
]
```

j. Push libraries to the device from the Ubuntu workstation. Run the following commands to side load the relevant libraries and binaries onto the device:

```
$ cd /opt/qcom/aistack/snpe/2.13.4.230831 on Ubuntu Workstation
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libcalculator_htp.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libcalculator.so             /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libhta_hexagon_runtime_snpe.so  /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libPlatformValidatorShared.so   /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
```

```
libSnpeDspV65Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeDspV66Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeHta.so                /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeHtpV68Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeHtpV73Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSNPE.so                   /usr/lib
$ adb push ./lib/hexagon-v65/unsigned/
libSnpeDspV65Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v65/unsigned/
libCalculator_skel.so          /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v66/unsigned/
libSnpeDspV66Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v68/unsigned/
libSnpeDspV68Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v69/unsigned/
libSnpeDspV69Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v73/unsigned/
libSnpeDspV73Skel.so           /usr/lib/rfsa/adsp
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-net-
run               /usr/bin
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-parallel-
run          /usr/bin
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-platform-
validator    /usr/bin
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-throughput-net-
run    /usr/bin
```

See *Qualcomm TensorFlow Lite Software Development Kit (TFLITE SDK) Tools Quick Start Guide* (80-50450-52) to do the following:

- Enable the TFLite GStreamer plugin
- Get the TFLite artifacts and update the relevant field (`Tflite_prebuilt_file`) in the above JSON file

5. Source the script to set up the environment:

```
$ cd <qim-sdk-workspace>/sdk-tools
$ source ./scripts/host/docker_env_setup.sh
```

6. Build the QIM SDK docker image:

```
$ qimsdk-docker-build-image ./targets/LE.PRODUCT.2.1.json
```

If the build set up fails, see Troubleshoot docker setup.

After successful completion, the following message is displayed:

```
"Status: Build image completed successfully !!!"
```

Running this step builds the QIM SDK as well.

7. Run the QIM SDK docker container. This starts the container with the tags provided in the JSON configuration file.

   ```
   $qimsdk-docker-run-container ./targets/LE.PRODUCT.2.1.json
   ```

   The following message is displayed:

   ```
   "Status: Docker qimsdk-ubuntu18-sdk-18 run successful !!!"
   ```

8. Attach the docker to the container:

   ```
   $ docker attach qimsdk-ubuntu18-sdk-18
   ```

   The docker shell is displayed and the following helper utilities are available in the bash environment:

```
SDK environment now set up; additionally you may now run devtool to perform development tasks.
Run devtool --help for further details.
Run add_external_layers --help to know about adding external workspace to eSDK
qimsdk-layers-build
    must be invoked to compile modified layers
qimsdk-layers-package
    must be invoked after invoking qimsdk-layers-build to package compiled recipes
qimsdk-device-prepare
    must be invoked to prepare device for pkg installation
qimsdk-device-sync-rel
    must be invoked to sync release packages with the device
qimsdk-device-sync-dbg
    must be invoked to sync debug packages with the device
qimsdk-device-packages-remove
    must be invoked to remove installed packages from the device
qimsdk-remote-sync-rel
    must be invoked to sync release packages with the remote target
qimsdk-remote-sync-dbg
    must be invoked to sync debug packages with the remote target
qimsdk-remote-sync-dev
    must be invoked to sync dev packages with the remote target
qimsdk-remote-sync-staticdev
    must be invoked to sync staticdev packages with the remote target
qimsdk-remote-packages-remove
    must be invoked to remove packages, installed by the remote target script
```

The QIM SDK is compiled, and the artifacts are ready to be deployed.

4.2 Connect device to host and deploy artifacts

After compilation, there are two mechanisms to connect the device to a host and deploy the QIM SDK artifacts.

- Device connected to local Linux host:

  A developer connects the device to a workstation and installs QIM SDK artifacts from the container directly on the device (QCS8550).

- Device connected to remote host:

  A developer connects the device to a remote workstation, and they can use the utilities provided by sdk-tools for Windows and Linux platforms to install QIM SDK artifacts to the device (QCS8550).

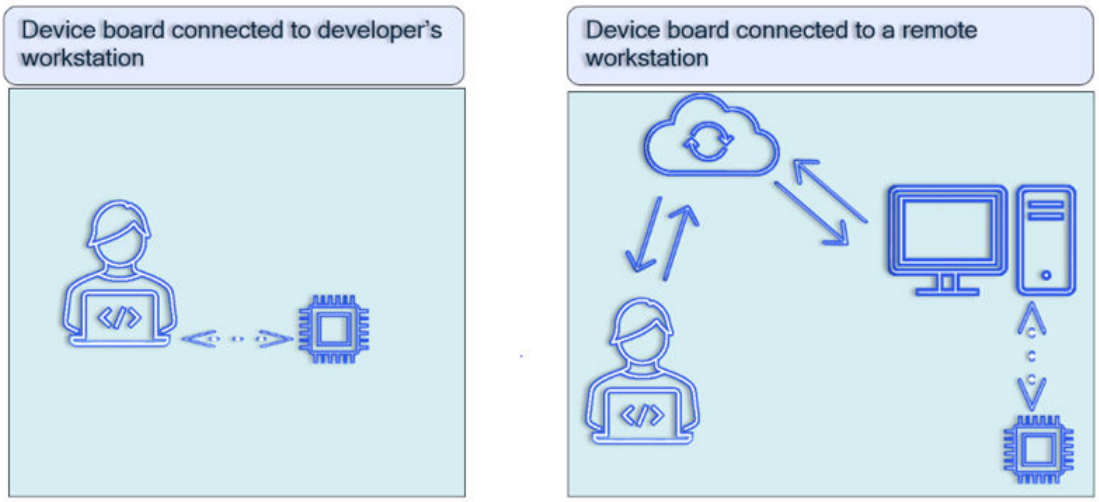
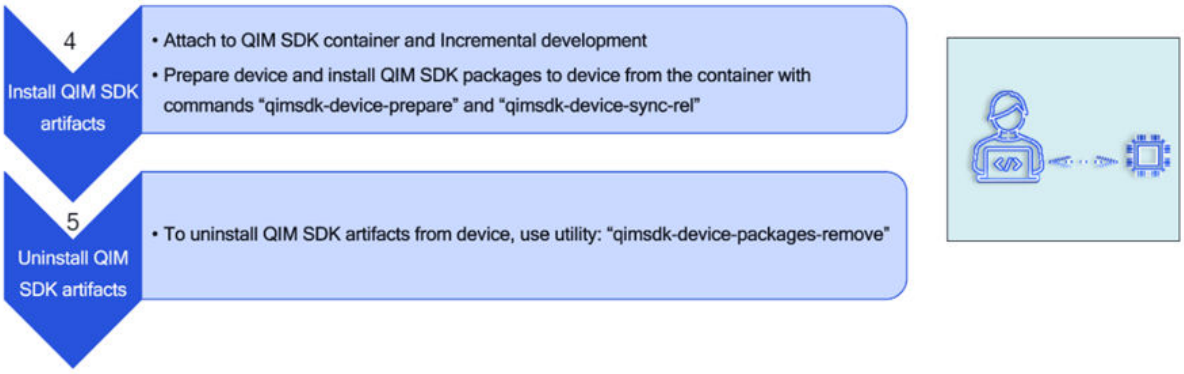


Figure 4-4 Connection of device board to developer and remote workstation

4.2.1 Connect device to workstation

The device is connected to the workstation and the QIM SDK container *can* access the device over USB/adb.

The figure shows the stages in the sequence of QIM SDK workflow:



.

1. Run the following commands to install the artifacts to the device:

```
$ qimsdk-device-prepare
$ qimsdk-device-sync-rel
```

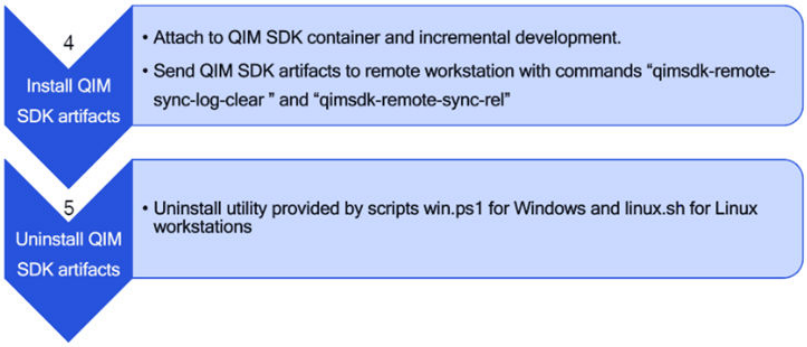
2. To uninstall the artifacts, run the following command:

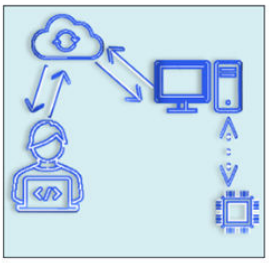
```
$ qimsdk-device-packages-remove
```

4.2.2 Connect device to remote machine

The device is connected to a remote machine, and the QIM SDK container *cannot* access the device over USB/adb.

The figure shows the stages in the sequence of the QIM SDK workflow:





1. Run the following commands to copy the artifacts to a remote machine:

```
$ qimsdk-remote-sync-log-clear
$ qimsdk-remote-sync-rel
```

NOTE The remote machine information is provided in the JSON configuration file.

2. Use the helper utilities in `<qim-sdk-workspace>/sdk-tools/scripts/local/` to install the artifacts on the device based on the operating system.

4.2.2.1 Install artifacts for Windows platform

The QIM SDK artifacts can be installed on the device based on the operating system of the remote machine.

For the Windows platform, do the following:

1. Copy the script `win.ps1` from `<qim-sdk-workspace>/sdk-tools/scripts/local/` to a remote Windows machine.
2. On PowerShell, use the following script:

```
PS  C:> .\win.ps1
PS C:> adb root
PS C:> adb disable-verity
PS C:> adb reboot
```

```
PS C:> adb wait-for-device
PS C:> adb root
PS C:> adb remount
PS C:> adb shell mount -o remount,rw /
PS C:> qimsdk-local-sync <path-to-package-dir>
```

4.2.2.2 Install artifacts for Linux platform

1. Copy the script `linux.sh` from `<qim-sdk-workspace>/sdk-tools/scripts/local` to remote Linux machine.
2. On bash, use the following script:

```
$ source linux.sh
$ adb root
$ adb disable-verity
$ adb reboot
$ adb wait-for-device
$ adb root
$ adb remount
$ adb shell mount -o remount,rw /
$ qimsdk-local-sync <path-to-package-dir>
```

4.3 Clean up docker image

After completing the developer workflow, the docker environment should be cleaned to free up the storage on the disk. Cleaning the docker removes the unused containers and images, thus freeing up the disk space.

Use the following commands to clean up the docker image:

1. Run the following command on the Linux workstation:

```
$ cd <qim-sdk-workspace>/sdk-tools
```

2. Stop the container:

```
$ qimsdk-docker-stop-container ./targets/LE.PRODUCT.2.1.json
```

3. Remove the container:

```
$ qimsdk-docker-rm-container ./targets/LE.PRODUCT.2.1.json
```

4. Remove the older docker images:

```
$ qimsdk-docker-cleanup
```

4.4 Troubleshoot docker setup

If the `qimsdk-docker-build-image` command returns a `No space left on device` message, then move the docker directory to `/local/mnt`.

Do the following to troubleshoot the setup:

1. Back up the existing docker files:

   ```
   $ tar -zcC /var/lib docker > /mnt/pd0/var_lib_docker-backup-$(date +
   %s).tar.gz
   ```

2. Stop the docker:

   ```
   $ service docker stop
   ```

3. Verify that no docker process is running:

   ```
   $ ps faux | grep docker
   ```

4. Check the docker directory structure:

   ```
   $ sudo ls /var/lib/docker/
   ```

5. Move the docker directory to a new partition:

   ```
   $ mv /var/lib/docker /local/mnt/docker
   ```

6. Make a symlink to the docker directory in the new partition:

   ```
   $ ln -s /local/mnt/docker /var/lib/docker
   ```

7. Ensure that the docker directory structure remains unchanged:

   ```
   $ sudo ls /var/lib/docker/
   ```

8. Start docker:

   ```
   $ service docker start
   ```

9. Restart all the containers after moving the docker directory.

4.5 Sync and build QIM SDK with Linux workstation

The QIM SDK workflow is also available within Linux workstation without having to install additional container packages like docker.

After the Linux workstation is installed with the prerequisite host packages for the QIM SDK workflow, the workflow for QIM SDK on the Linux workstation can be initiated by sourcing the `host_env_setup.sh` script from the `sdk-tools/scripts/host` folder.

To sync and build QIM SDK, do the following

1. Create a directory on the host file system to sync the QIM SDK workspace. For example:

   ```
   $mkdir <qim-sdk-workspace>
   $cd <qim-sdk-workspace>
   ```

2. Fetch the QIM SDK source code from CodeLinaro:

```
$ repo init -u https://git.codelinaro.org/clo/le/sdkqim/qim/manifest.git
-- repo-branch=qc/stable --repo-url=git://git.quicinc.com/tools/repo.git -
m QIM.SDK.1.0.0.r1-01100-QIM.0.xml -b release && repo sync -qc --no-tags -
j8
```

3. Edit the JSON configuration file present in `<qim-sdk-workspace>/sdk-tools/targets/LE.PRODUCT.2.1` with the following entries:

```
{
"Image_OS": "ubuntu18",
"Additional_tag": "sdk-18",
"eSDK_shell_file": "Absolute path to Platform extensible SDK file on Linux
Workstation",
"Base_Dir_Location": "path-to-the-directory-where-the-project-is-
initialized-applies-only-for-Linux-Workstation(host)-build-variant",
"Tflite_prebuilt_file": "absolute path to tf-lite-prebuilt-dev-tar-file
generated from TfLite SDK",
"Acceleration_engines":
[
{
"Acceleration_engine":"name of acceleration engine 1",
"Acceleration_engine_path":"absolute path to Acceleration Engine SDK 1"
},
{
"Acceleration_engine":"-", "Acceleration_engine_path":"-"
}
],
"Host_dir_mounted_in_container": "<Directory on Host that will be mapped
into QIMSDK container to get artifacts to host (not applicable to Linux
Workstation workflow))",
"Deploy_URL":”<not applicable>",
"Deploy_dev_URL": "<not applicable>",
"Deploy_QIMSDK_Artifacts_URL":"Directory on Host where all QIMSDK
artifacts archive will be made available",
"Gst_plugins_qti_oss_dependencies":["cairo", "ffmpeg", "gdk-pixbuf",
"liba52", "libgudev", "lame", "librsvg", "libtheora", "libusb1",
"libwebp", "mpg123", "orc", "sbc", "speex", "taglib", "vulkan-loader"]
}
```

For more information on the entries mentioned in the json configuration file, see the `Host.md` readme file present at `<qim-sdk-workspace>/sdk-tools/`.

Enable AI inference plug-ins:

Qualcomm Neural Processing SDK

QIM SDK workflow can generate the QIM SDK plugin for Qualcomm Neural Processing SDK (Formerly known as SNPE).

To build the SNPE QIM SDK plug-in, download the SNPE SDK needs to be downloaded and path to SNPE sdk on Linux workstation needs to be provided in configuration JSON file.

a. To download Qualcomm Package Manager 3 for Ubuntu, clickhttps://qpm.qualcomm.com/, and click **Tools**.

b. In the left pane, in the **Search Tools** field, type `QPM`. From the **System OS** list, select **Linux**.

   The search results display a list of Qualcomm Package Managers.

c. Select Qualcomm Package Manager 3 and download the Linux debian package.

d. To install Qualcomm Package Manager 3 for Linux, use the following command:

```
$ dpkg -i --force-overwrite /path/to/
QualcommPackageManager3.3.0.83.1.Linux-x86.deb
```

e. To download Qualcomm Snapdragon Neural Processing Engine SDK, clickhttps://qpm.qualcomm.com/, and click **Tools**.

f. In the left pane, in the **Search Tools** field, type `ai stack`. From the **System OS** list, select **Linux**.

   The search results display a list of Qualcomm Package Managers..

g. Click Qualcomm® Neural Processing SDK, and then download the .qik package with Linux version 2.13.0.

h. To install Qualcomm® Neural Processing SDK on Ubuntu workstation, run the following commands:

```
$ qpm-cli --login <username>
$ qpm-cli --license-activate qualcomm_neural_processing_sdk
$ qpm-cli --extract <full path to downloaded .qik file>
```

   After the extraction is successful, the following message is displayed."`/opt/qcom/aistack/snpe/2.13.4.230831`"

i. Add the entries into `LE.PRODUCT.2.1.json` file. For example:

```
"Acceleration_engines":
[
{"Acceleration_engine":"snpe",
"Acceleration_engine_path":”/opt/qcom/aistack/snpe/2.13.4.230831"
},
]
```

j. Push libraries to the device from the Ubuntu workstation. Run the following commands to side load the relevant libraries and binaries onto the device:

```
$ cd /opt/qcom/aistack/snpe/2.13.4.230831 on Ubuntu Workstation
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libcalculator_htp.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libcalculator.so             /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libhta_hexagon_runtime_snpe.so  /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libPlatformValidatorShared.so   /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
```

```
libSnpeDspV65Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeDspV66Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeHta.so                /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeHtpV68Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSnpeHtpV73Stub.so         /usr/lib/
$ adb push ./lib/aarch64-oe-linux-gcc11.2/
libSNPE.so                   /usr/lib
$ adb push ./lib/hexagon-v65/unsigned/
libSnpeDspV65Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v65/unsigned/
libCalculator_skel.so          /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v66/unsigned/
libSnpeDspV66Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v68/unsigned/
libSnpeDspV68Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v69/unsigned/
libSnpeDspV69Skel.so           /usr/lib/rfsa/adsp
$ adb push ./lib/hexagon-v73/unsigned/
libSnpeDspV73Skel.so           /usr/lib/rfsa/adsp
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-net-
run               /usr/bin
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-parallel-
run          /usr/bin
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-platform-
validator    /usr/bin
$ adb push ./bin/aarch64-oe-linux-gcc11.2/snpe-throughput-net-
run    /usr/bin
```

See *Qualcomm TensorFlow Lite Software Development Kit (TFLITE SDK) Tools Quick Start Guide* (80-50450-52) to do the following:

- Enable the TFLite GStreamer plugin
- Get the TFLite artifacts and update the relevant field (`Tflite_prebuilt_file`) in the above JSON file

4. Add the following lines in the `scripts/host/host_env_setup.sh` script within the `qimsdk-host-env-setup()` function.

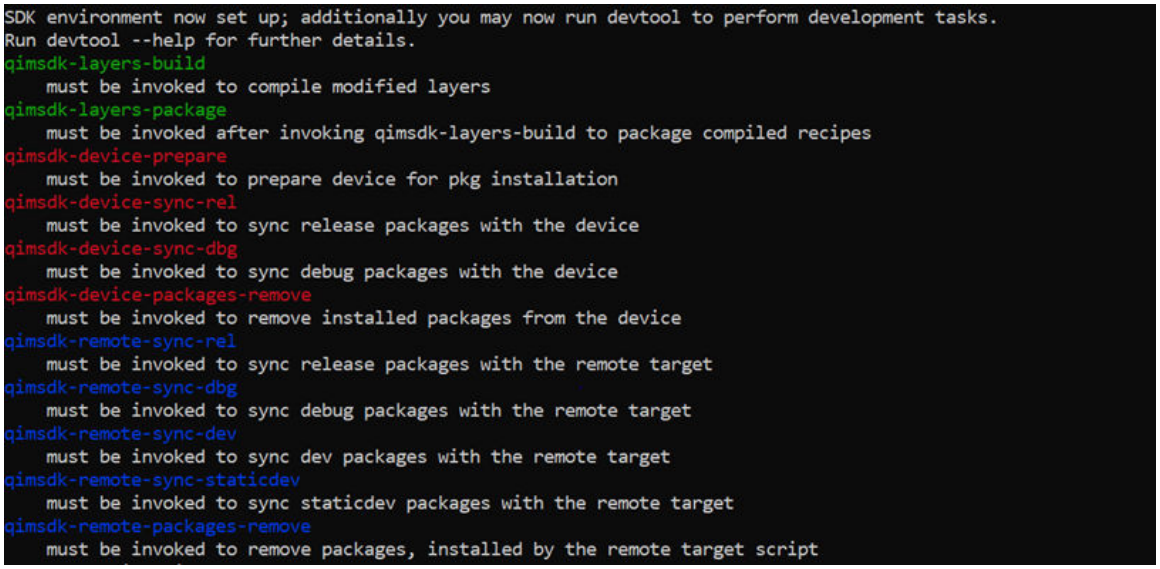
```
export QIMSDK_ESDK_BASE_DIR=${QIMSDK_BASE_DIR}/esdk
export QIMSDK_WORK_DIR=${QIMSDK_BASE_DIR}/work
export QIMSDK_SCRIPTS=${QIMSDK_BASE_DIR}/scripts
```

5. Set up the workflow for Linux workstation QIM SDK by sourcing the `host_env_setup.sh` file.

```
$ source sdk-tools/scripts/host/host_env_setup.sh
```

6. Use the following command to resolve any dependencies and run the QIM SDK workflow to build the QIM SDK packages:

   ```
   $ qimsdk-setup   targets/LE.PRODUCT.2.1
   ```

   The following message is displayed after the `qimsdk-setup` utility runs successfully.

   ```
   All layers packaged successfully !!!

   IM SDK setup completed !!!
   ```

7. To transfer the available QIM SDK packages to `Deploy_QIMSDK_Artifacts_URL`, run the `qimsdk-host-sync-artifacts-all` utility.

   ```
   $ qimsdk-host-sync-artifacts-all targets/LE.PRODUCT.2.1.json
   ```

8. Extract the `packages.zip` into a folder. To install the artifacts from QIM SDK packages, see Install artifacts for Windows platform and Install artifacts for Linux platform.

4.6 Troubleshoot host setup

The host setup issues could be related to the missing packages that must be installed on the Linux host machine before building QIM SDK.

Follow the messages displayed on the prompt to troubleshoot the host setup.

5 Build QIM SDK incrementally

If you are building QIM SDK for the first time, see Build QIM SDK – Developer workflow. The same build environment can be reused for incremental plugins and application development.

The helper utilities (within the container) mentioned in the figure are available to developers to compile modified applications and plugins.

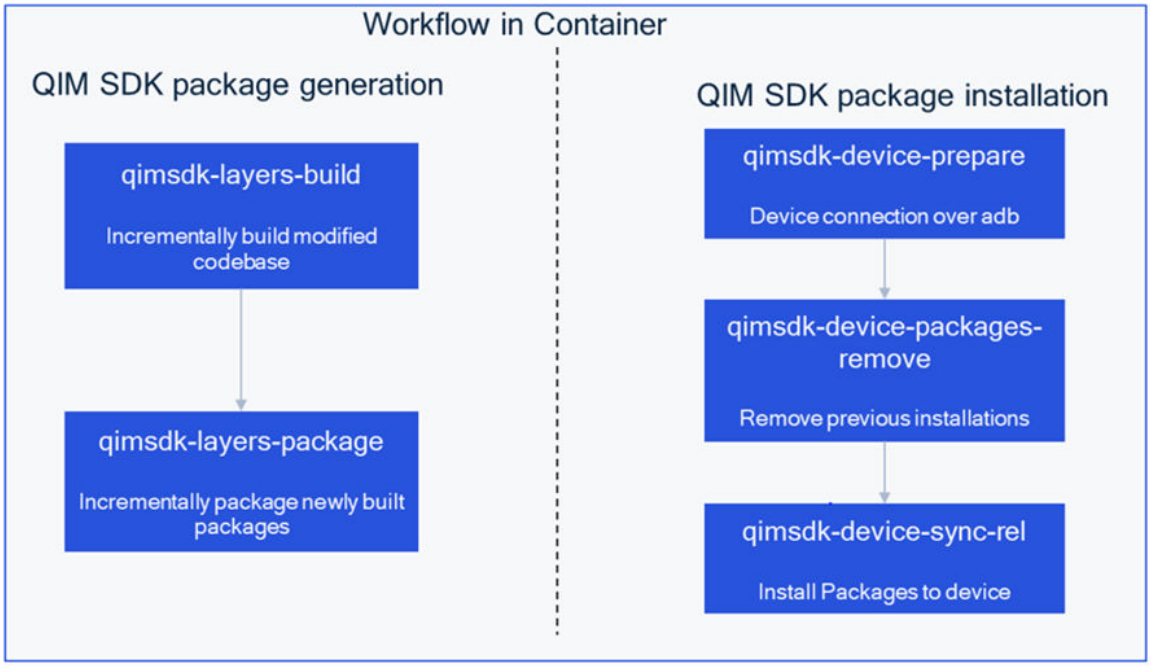


Figure 5-1 Workflow in container

After the code changes are completed in the code directory, run the following commands:

1. Compile modified code:

   ```
   $ qimsdk-layers-build
   ```

2. Package compiled code:

   ```
   $ qimsdk-layers-package
   ```

3. Sync release packages with the host file system:

   ```
   $ qimsdk-remote-sync-log-clear && qimsdk-remote-sync-rel
   ```

4. Sync dev packages with the host file system:

   ```
   $ qimsdk-remote-sync-log-clear && qimsdk-remote-sync-dev
   ```

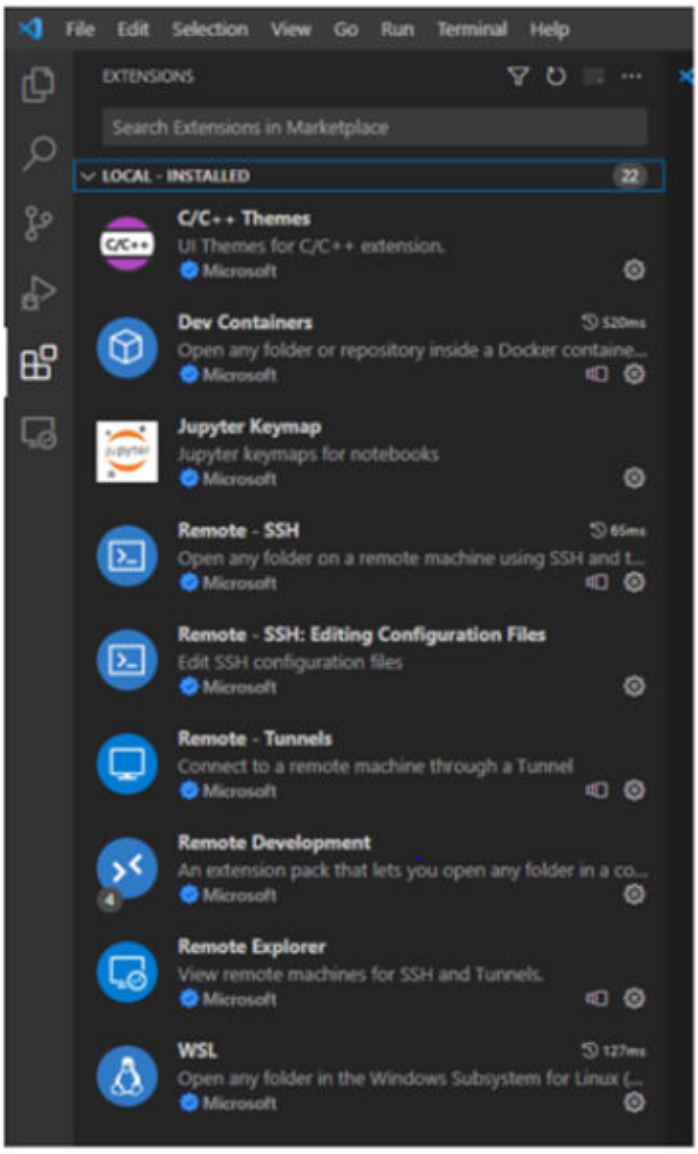
The compiled artifacts are found at `Host_dir_mounted_in_container` folder mentioned in the JSON file, which can be copied to any directory.

6 Develop applications using Visual Studio Code

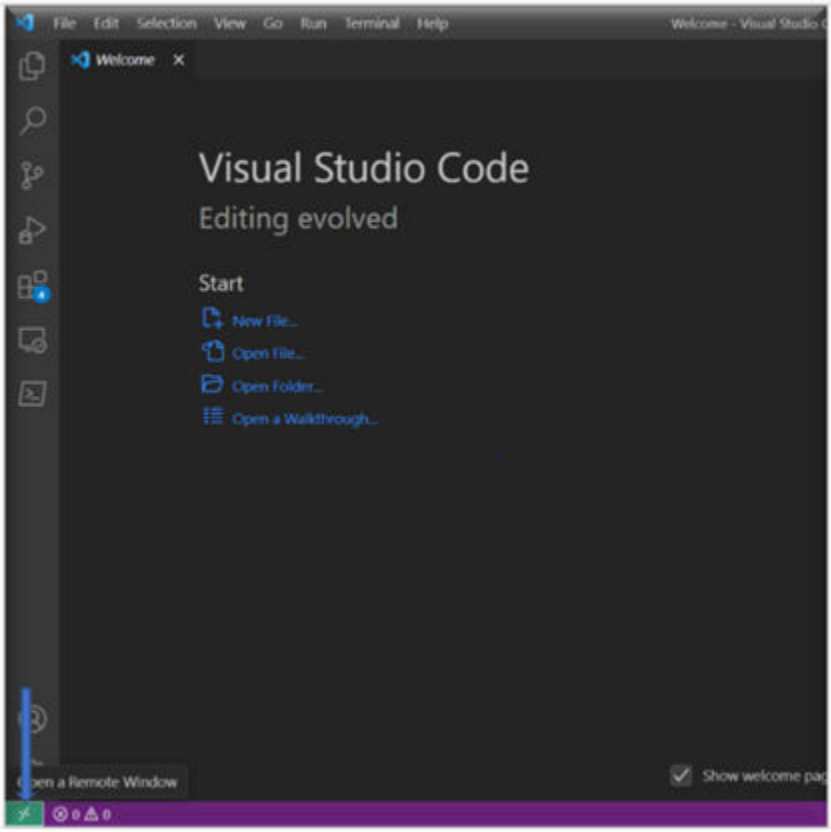
Visual Studio Code is a code editor that support development operations on a remote workstation using QIM SDK.

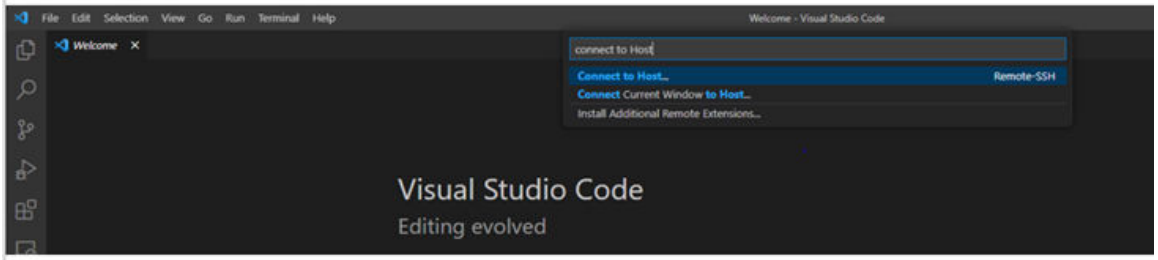
For more information about Visual Studio Code, see https://code.visualstudio.com/.

1. Set up Visual Studio Code with extensions related to remote work profiles.

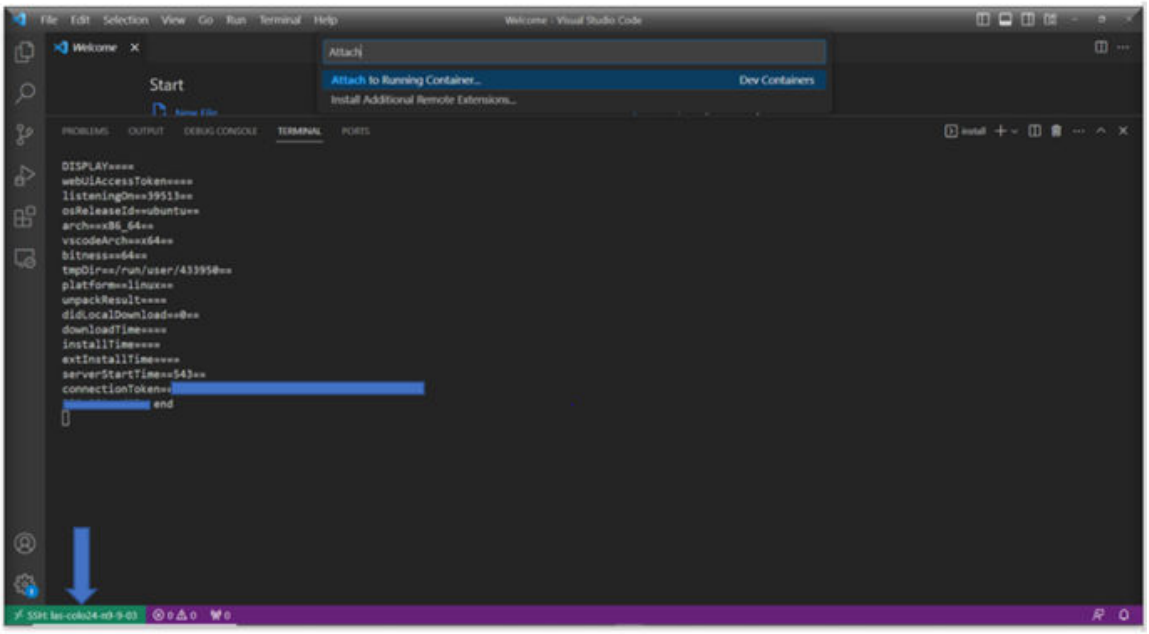


2. Connect to remote workstation.

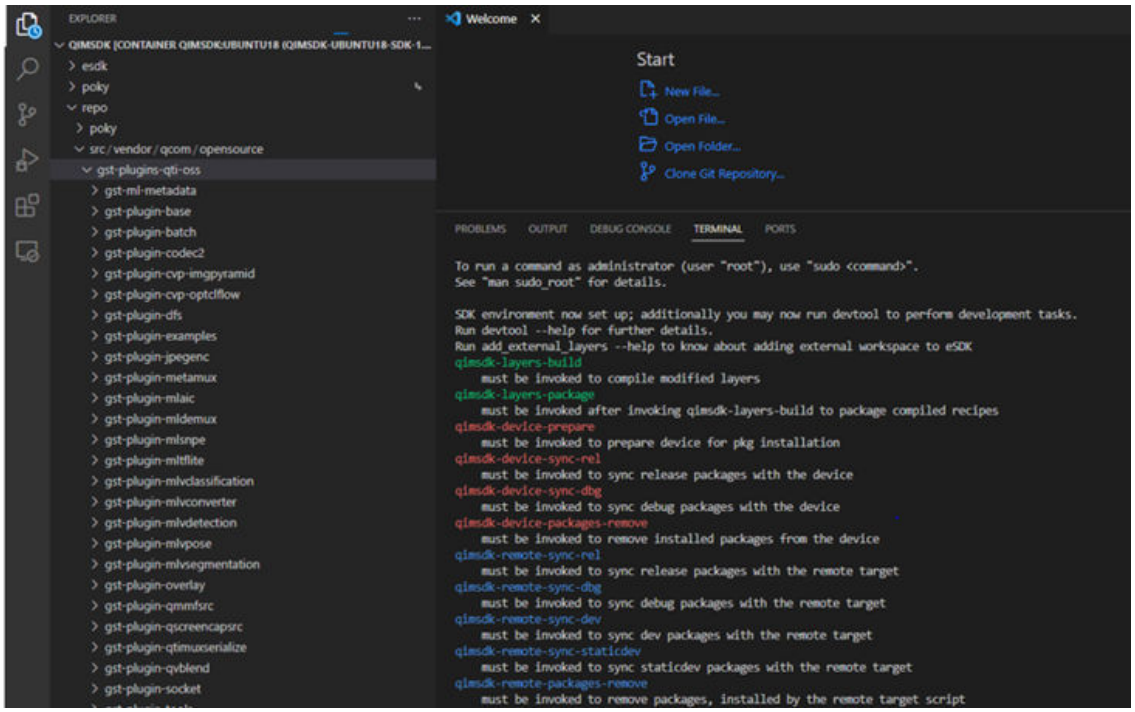


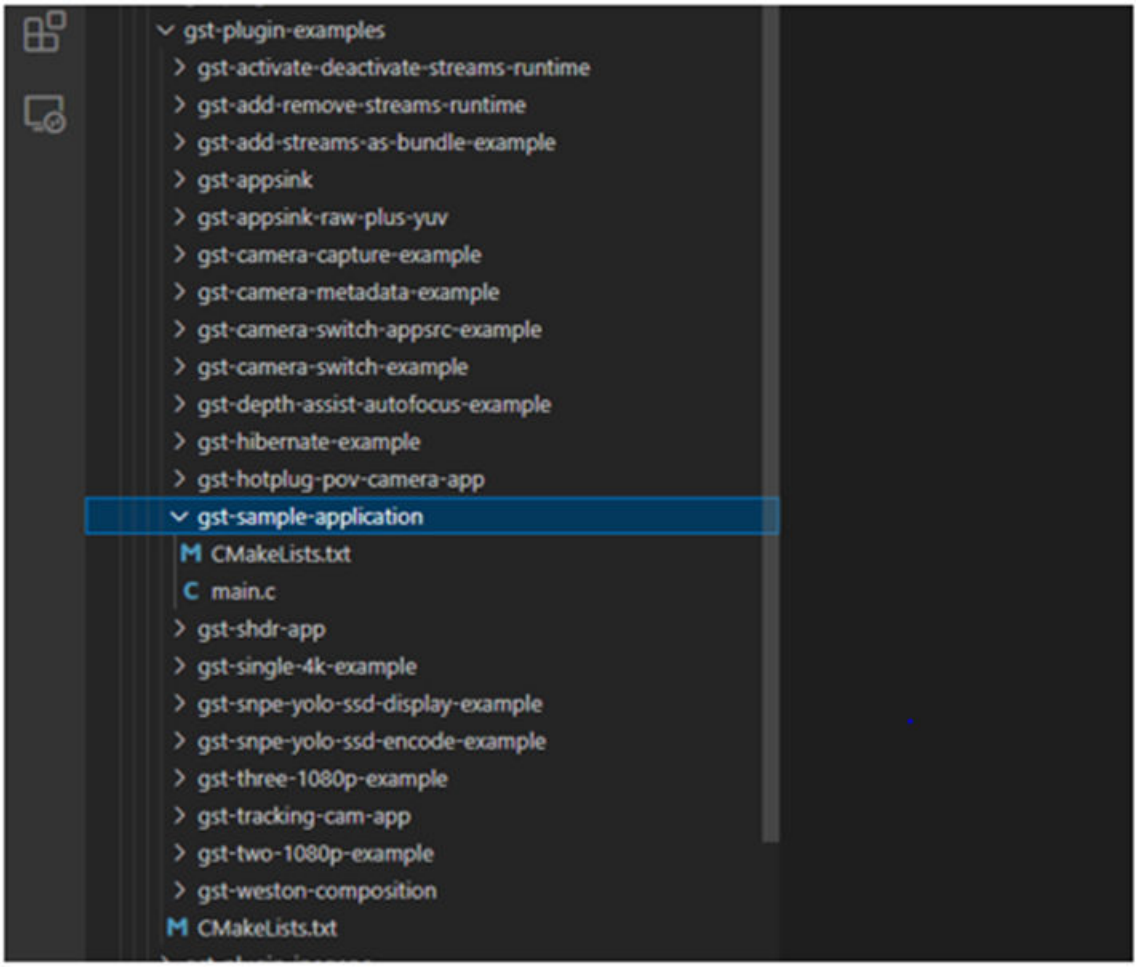


3. Connect to the QIM SDK container on the remote workstation.



4. Open the `gst-plugin-example` directory to add and develop new applications.





5. See Build QIM SDK – Developer workflow to generate an example application as part of `ipk` for the `gst-plugin-examples` package.

   For more information on QIM SDK plug-ins and applications, see *QCS8550.LE.1.0 Qualcomm Intelligent Multimedia SDK (QIM SDK) Reference* (80-50450-50).

LEGAL INFORMATION

Your access to and use of this document, along with any specifications, reference board files, drawings, diagnostics and other information contained herein (collectively this "Documentation"), 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 Documentation and shall immediately destroy any copy thereof.

1) Legal Notice.

This Documentation is being made available to You solely for Your internal use with those products and service offerings of Qualcomm Technologies, Inc. ("Qualcomm Technologies") and its affiliates described in this Documentation, and shall not be used for any other purposes. This Documentation may not be altered, edited, or modified in any way without Qualcomm Technologies' prior written approval. Unauthorized use or disclosure of this Documentation 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 licensors for any such unauthorized uses or disclosures of this Documentation, in whole or part.

Qualcomm Technologies, its affiliates and licensors retain all rights and ownership in and to this Documentation. No license to any trademark, patent, copyright, mask work protection right or any other intellectual property right is either granted or implied by this Documentation 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 Documentation.

THIS DOCUMENTATION 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 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 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 DOCUMENTATION.

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

Technical data specified in this Documentation 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 Documentation is an offer to sell any of the components or devices referenced herein.

This Documentation 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 or the *Qualcomm Privacy Policy* referenced on www.qualcomm.com, 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) executed by You and Qualcomm Technologies or a Qualcomm Technologies affiliate with respect to Your access to and use of this Documentation, 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 Documentation may be trademarks or registered trademarks of their respective owners.

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